NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

MOTION PLANNING AND DYNAMIC CONTROL OF THE NOMAD 200 MOBILE ROBOT IN A LABORATORY ENVIRONMENT

by

Ko-Cheng Tan

June, 1996

Advisor: Xiaoping Yun
Second Reader: Isaac Kaminer

Approved for public release; distribution is unlimited.

19960920 023

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE MOTION PLANNING AND DYNAMIC CONTROL OF THE NOMAD 200 MOBILE ROBOT IN A LABORATORY ENVIRONMENT		5. FUNDING NUMBERS
6. AUTHOR(S) Ko-Cheng Tan		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

  Motion planning and control of a Nomad 200 mobile robot are studied in this thesis. The objective is to develop a motion planning and control algorithm that is able to move the robot from an initial configuration (position and orientation) to a goal configuration in a typical laboratory environment. The robot must be able to avoid unknown static (e.g., walls and tables) and dynamic (e.g., people) obstacles. Dubin's algorithm finds the shortest path connecting two configurations in an obstacle-free environment, but it is not able to avoid obstacles present in the environment. The potential field algorithm is effective in avoiding unknown obstacles, but it has the local minimum problem and does not consider the orientation of a mobile robot. A modified potential field algorithm is first developed. The algorithm overcomes local minima in a typical laboratory environment. The modified potential field algorithm is then combined with Dubin's algorithm to incorporate orientation into motion planning. The combined algorithm is able to avoid static and dynamic obstacles and achieve position and orientation requirements. Simulation and physical experiment results are presented to demonstrate the effectiveness of the algorithm.

14. SUBJECT TERMS :Potential Field Method, Nomad, Linearization Feedback			15. NUMBER OF PAGES 90
			16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST
QUALITY AVAILABLE. THE
COPY FURNISHED TO DTIC
CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO
NOT REPRODUCE LEGIBLY.

MOTION PLANNING AND DYNAMIC CONTROL OF THE NOMAD 200 MOBILE ROBOT IN A LABORATORY ENVIRONMENT

Ko-Cheng Tan
Lieutenant Commander, Republic of China Navy
B.S., Chinese Naval Academy - 1984

Submitted in partial fulfillment
of the requirements for the degree of

**MASTER OF SCIENCE
IN
AERONAUTICAL ENGINEERING**

from the

**NAVAL POSTGRADUATE SCHOOL
June 1996**

Author: _____
Ko-Cheng Tan

Approved by: _____
Xiaoping Yun, Advisor

_____
Isaac Kaminer, Second Reader

_____
Daniel Collins, Chairman
Department of Aeronautics and Astronautics

iii

ABSTRACT

Motion planning and control of a Nomad 200 mobile robot are studied in this thesis. The objective is to develop a motion planning and control algorithm that is able to move the robot from an initial configuration (position and orientation) to a goal configuration in a typical laboratory environment. The robot must be able to avoid unknown static (e.g., walls and tables) and dynamic (e.g., people) obstacles. Dubin's algorithm finds the shortest path connecting two configurations in an obstacle-free environment, but it is not able to avoid obstacles present in the environment. The potential field algorithm is effective in avoiding unknown obstacles, but it has the local minimum problem and does not consider the orientation of a mobile robot. A modified potential field algorithm is first developed. The algorithm overcomes local minima in a typical laboratory environment. The modified potential field algorithm is then combined with Dubin's algorithm to incorporate orientation into motion planning. The combined algorithm is able to avoid static and dynamic obstacles and achieve position and orientation requirements. Simulation and physical experiment results are presented to demonstrate the effectiveness of the algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

I.   INTRODUCTION

Benefited from technical innovations in computer and sensor technologies, robots are becoming more intelligent and are widely used in many applications. Mobile robots in particular have attracted a lot of attentions in the recent years and represent a fast growing area in robotics. There are three fundamental questions in mobile robot research: (i) where am I; (ii) where am I going; and (iii) how will I go there.

The first question is concerned with navigation of mobile robots. The second question addresses issues of task planning. For example, a mobile robot is given a task of bring a cup of coffee to someone. The task planner must decide where to go to find a nearby coffee machine. The third question deals with motion planning and control of mobile robots, which is the topic of this thesis.

While a task planner decides where a robot must go (i.e., a goal position) based on task requirements, a motion planner is responsible for selecting a path from the current position of the robot to the goal position. Many motion planning algorithms have been developed in the literature. They may be classified in various ways. In terms of obstacles, Dubin [Ref. 5] developed an algorithm for connecting an initial configuration (position plus orientation) to a final configuration with the shortest distance in an obstacle-free environment. The algorithm uses a piece of arc at the beginning and at the end, connecting two arcs by a straight line. Then there are motion planning algorithms for environments with obstacles. While some algorithms assume static obstacles, others allow moving obstacles. Motion planning algorithms may also be divided into local algorithms and global algorithms. Global algorithms require that the information on the environment (the size, shape, and location of obstacles) be given to the motion planner a priori. On the other hand, local algorithms make decisions based on local information, typically gathered by onboard sensors. The Voronoi diagram, visibility graph, and cell decomposition methods are examples of global motion planning algorithms [Ref. 4]. The potential field method is an example of local algorithms [Ref. 4].

In the potential field method, a mobile robot is considered to be subjected to an artificial potential force. The potential force has two components: attractive force and repulsive force. The goal position produces an attractive force which makes the mobile robot move towards it. Obstacles generate a repulsive force, which is inversely proportional to the distance from the robot to obstacles and is pointing away from obstacles. Although the potential field method is an effective planning algorithm, it has a drawback of local minimum and does not consider the orientation of a robot.

The objective of this thesis is to develop a planning and control algorithm that overcomes the local minimum problem in a typical laboratory environment and include orientation of mobile robots in motion planning. While Dubin's planning algorithm includes orientation, it does not consider obstacles. Other planning algorithms that consider obstacles treat a robot as a point. These planning algorithms are developed to move a robot from one position to another without regard to robot's orientation. In many applications, orientation is as important as position motion. Vehicle parking is clearly an example. In military applications, orientation of an unmanned vehicle is critical for proper firing direction.

In this thesis, a modified potential field algorithm is first developed. The algorithm overcomes local minima in a typical laboratory environment. The modified potential field algorithm is then combined with Dubin's algorithm to incorporate orientation into motion planning. The combined algorithm is able to avoid static and dynamic obstacles and achieve position and orientation requirements. In designing feedback controller of the Nomad 200 mobile robot, a dynamic, nonlinear model is developed. Feedback linearization technique is utilized to linearize the nonlinear model. A linear feedback is finally designed for the linearized system to achieve smooth motion requirements.

The thesis is organized as follows. The next chapter reviews the background and previous work relevant to this study. The main results of this thesis are presented in Chapter III. The first section of this chapter is devoted to the dynamic modeling and feedback control of the Nomad 200 mobile robot. The second section presents the modified potential field algorithm and the combined algorithm. The third section describes simulation results

2

as well as results from physical experiments. The overall results are summarized in Chapter IV.

II. BACKGROUND AND RELATED WORK

A.   INTRODUCTION TO NOMAD 200 MOBILE ROBOT

The Nomad 200 is an integrated mobile robot intended for research. The robot is composed of two major components including robot itself (Figure 1) and a remote control host computer system. The host computer is based on a Sun workstation running UNIX operating system and provides the robot's remote control by radio and simulation function (Figure 2). The robot itself is made up of three major hardware units: processing unit, mechanical unit and sensor unit. The robot has two working modes: stand along mode and host control mode.
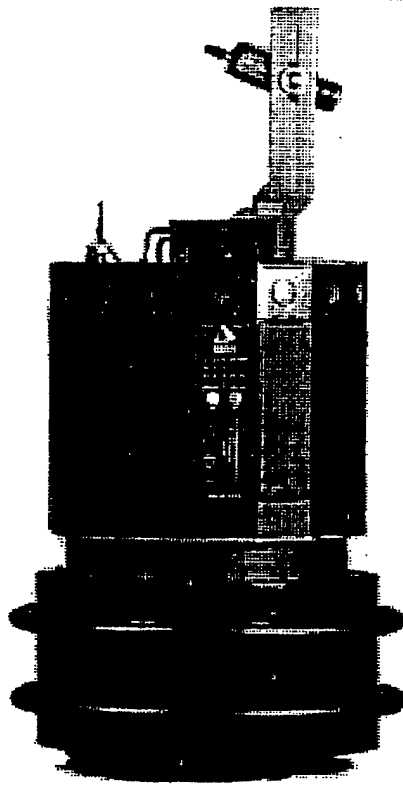
Figure 1. Nomad 200 Robot [Ref. 1]
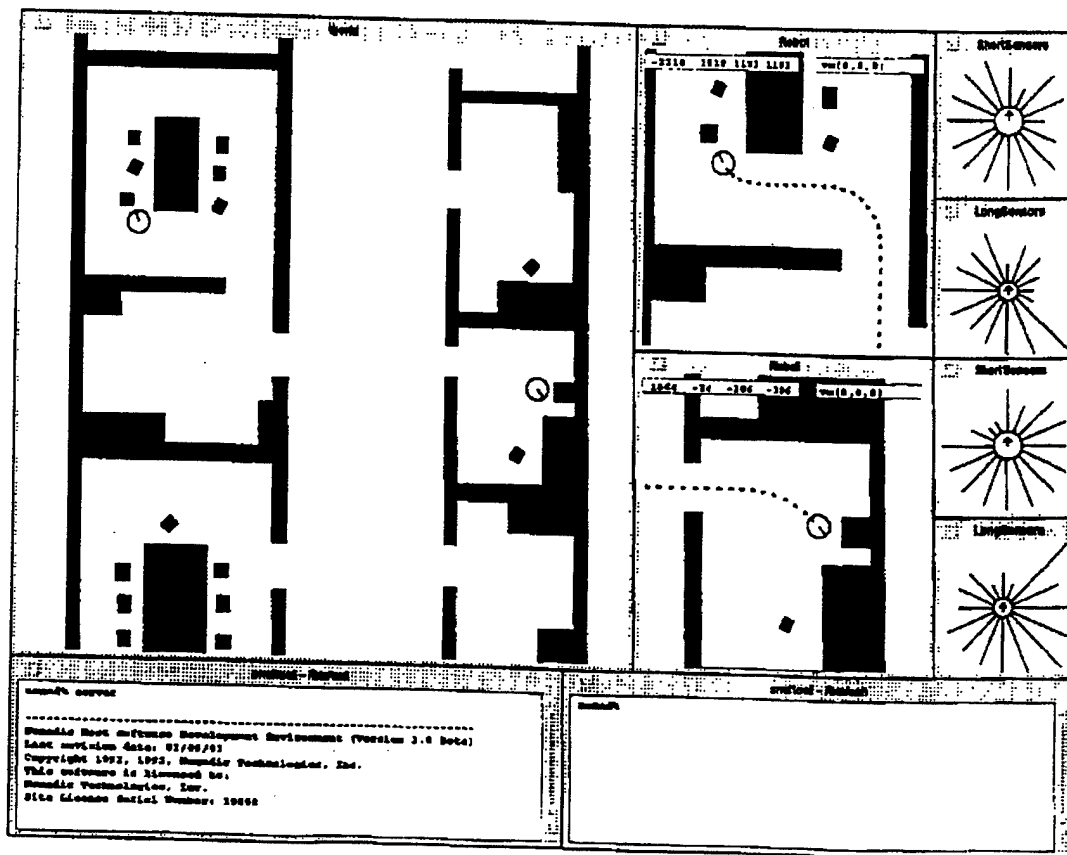
Figure 2. Nomad Software Simulation Environment

The Nomad 200 processing unit is a multiprocessor system (Figure 3) which operates the motor control, sensor control, information process and communication with the host computer. The multiprocessor system utilizes memory share technique, and communicates with each other on an Industry Standard Architecture (ISA) bus. The master processor is a 486 CPU. The sensor interface is composed of Motorola MC68HC11 16 MHz controllers. The motor interfacing is performed by a Motorola 68008/ASIC control system. The controller executes the master processor's command and responds with the required information.

6
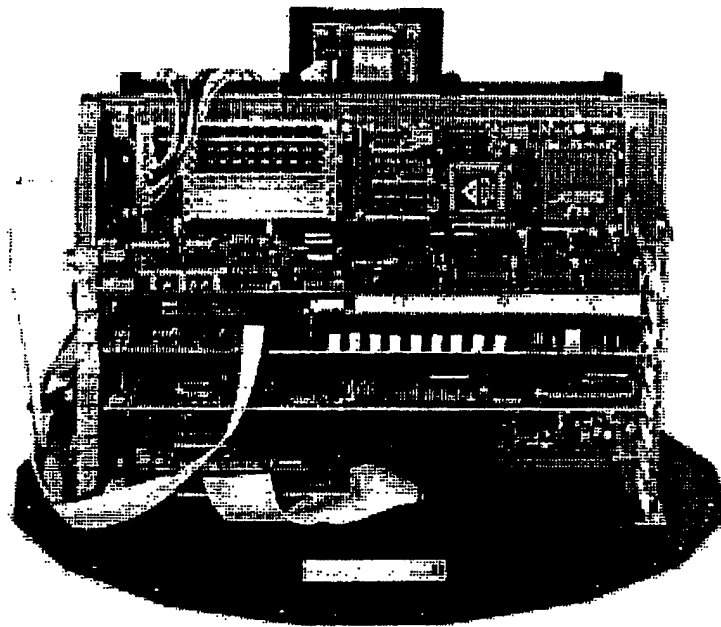
Figure 3. Multi-processor System [Ref. 1]

The sensor unit is equipped with three kinds of sensors: tactile, infrared and sonar sensor. The tactile sensor (Figure 4) is a 20 channel tactile system consisting of 20 independent pressure sensitive sensors (switches). The 20 switches are organized in two rings with ten switches in each ring. The switches on the top and bottom rings are interleaved to provide 360 degree coverage with an 18 degree resolution. Each sensor has approximately an eight ounce sensitivity.

The infrared sensor (Figure 5) is a 16 channel reflective intensity based infrared ranging system. The sensors can give range information up to 35 inches, under the proper conditions. Range to the object is determined by the intensity of the light from the emitter reflected back to the detectors of an object.

The sonar sensor (Figure 6) is a 16 channel sonar ranging system. The sonar sensors can give range information from 5 inches to 255 inches with a 1% accuracy over the entire range. The sensors use standard Polaroid transducers. Each transducer has a beam width of

25 degrees. The circumference of the robot is covered by 16 sensors. All three types of sensors use Motorola 68HC11 micro controllers.

Pressure Sensitive Switches
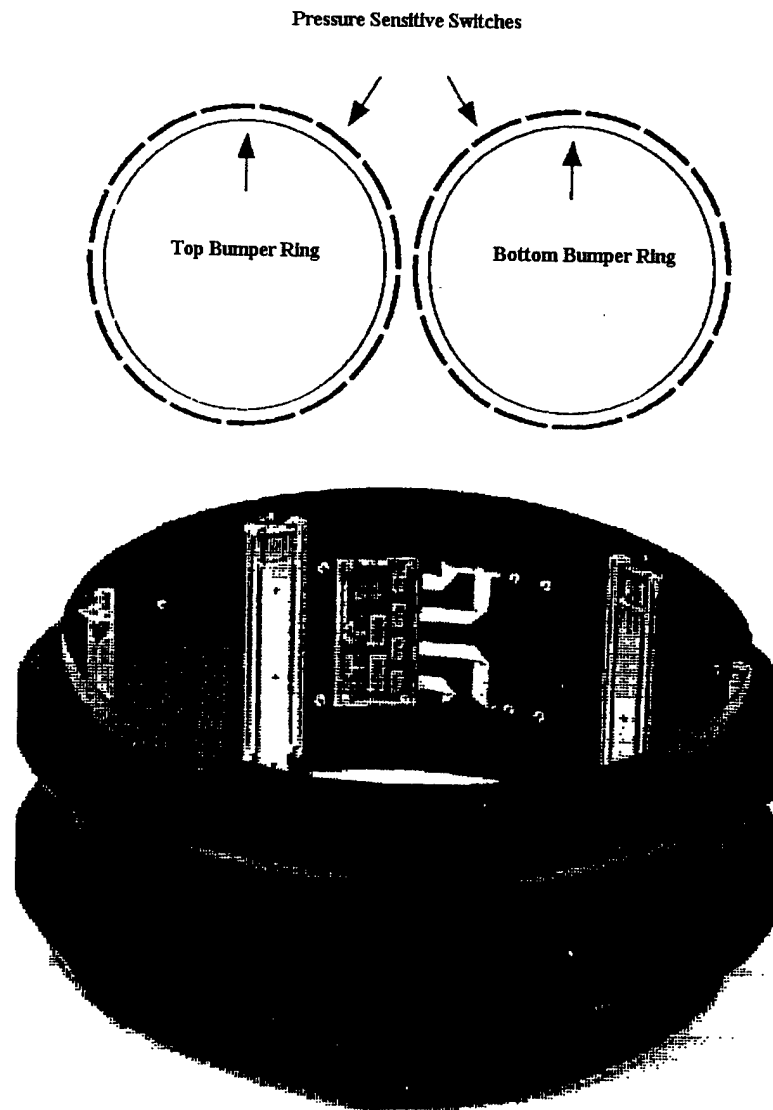
Figure 4. Tactile Sensor System [Ref. 1]
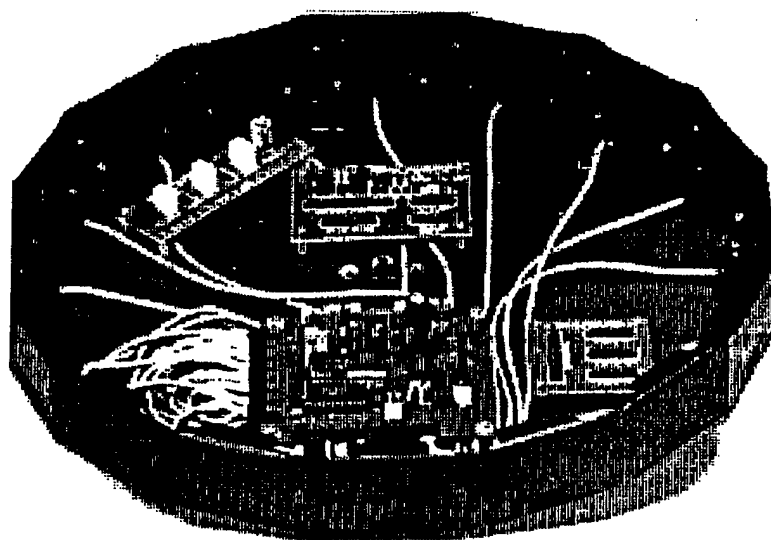
Figure 5. Infrared Sensor System [Ref. 1]
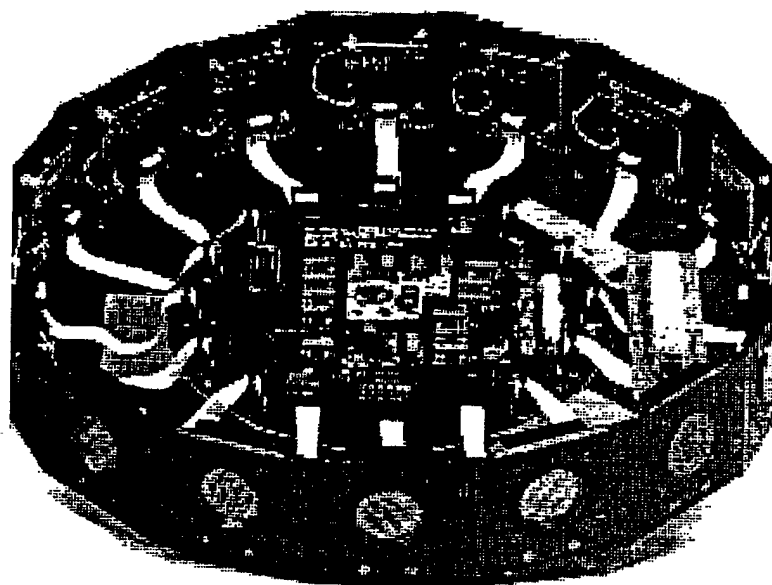
Figure 6. Sonar Sensor System [Ref. 1]

9

The mechanical unit of Nomad 200 is composed of the mobile base, turret and drive system. The turret can rotate independently of mobile base. The infrared and sonar sensor are mounted on the turret. The bumper sensor is installed on the mobile base. The mobile base translates according to the alignment of the driving wheels. Ideally, the base will not rotate once the coordinates are set. The alignment of the base should never change. However, a rough surface may affect the alignment. The drive system is a three servo motor system. One motor controls the angular position of the turret, and the other two motors drive a three wheel's translate and rotate separately (all three wheels, Figure 7, are synchronous, non-holonomic). The three wheels translate and rotate simultaneously. The robot's position and orientation are calculated by integrating the wheel's motion over time. This driving system will provide the robot a maximum translation speed of 20 inches per second and a maximum rotational speed of 60 degrees per second. The entire mechanical unit has a radius of 18 inches and a height of 30 inches.
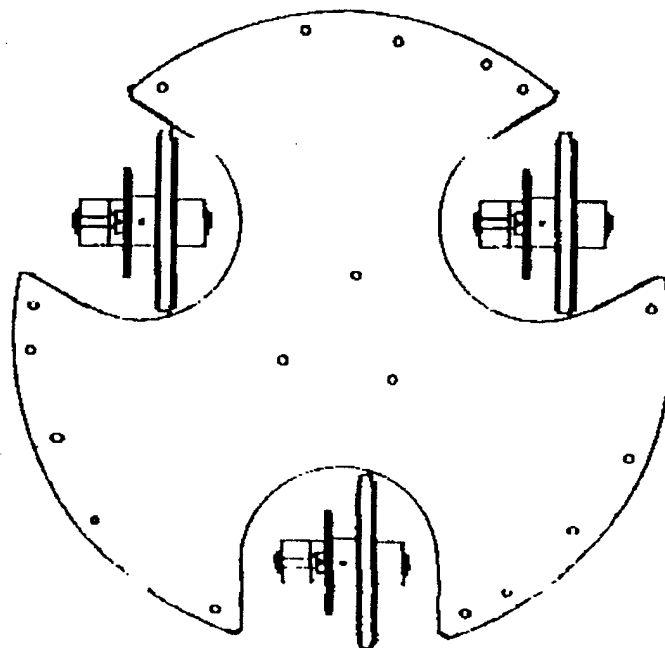
Figure 7. Nomad Wheel System [Ref. 1]

10

B.   LAGRANGE EQUATIONS

This section reviews Lagrange equations that will be used to derive equations of motion of Nomad 200 in Chapter III, Section A, Subsection 1. The approach of Lagrange equations involves finding a set of generalized coordinates, which completely parameterize the configuration space of a system, and generate the corresponding generalized forces. Based on these general coordinates, we can build up the system's dynamics model.

Any set of coordinates which can express the configuration of a system are generalized coordinates. For example, the system's dynamics of a two-link planar pendulum (Figure 8) can be expressed by a Lagrange equation using general coordinates $\{\theta_1, \theta_2\}$. Since these two are independent, the system has two degrees of freedom.
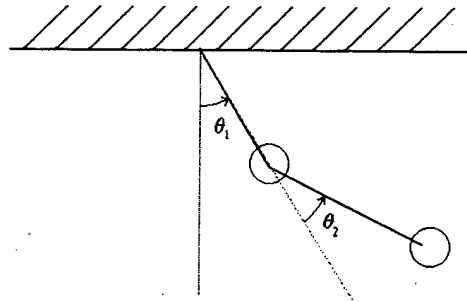
Figure 8. Two Planar Pendulum [Ref. 7]

Generally, the discussion on constraints is divided into holonomic and non-holonomic constraints. Holonomic constraints can be expressed as:

$$\varsigma_j\left(g_1, g_2 \ldots, g_n, t\right) = 0 \qquad (1)$$

where $j = 1 \ldots m$, and g_i is the general coordinate.

If the constraints cannot be expressed in the above form, and it can be expressed in terms of the differential form of general coordinates with function of time, they are non-

holonomic constraints. The D'Alembert's principle states that: "the force which is generated by the constraints does no work on the system" [Ref. 2]. This statement gives us the result:

$$\Gamma \bullet \dot{q} = (\frac{\partial h^T}{\partial q} \lambda) \bullet \dot{q} = (A^T(q) \lambda) \bullet \dot{q} = 0 , \tag{2}$$

where Γ is the linear combination of non-holonomic constraints, and λ is the vector associated to the constraint forces.

The Lagrange equation is formulated in terms of the kinematics energy and potential energy. It can be expressed as follows:

$$\frac{d}{dt} \left(\frac{\partial}{\partial \dot{g}_k} L \right) - \frac{\partial}{\partial g_k} L + A^T(q) \lambda - \gamma = 0 , \tag{3}$$

where $L = T - V$, T is kinetic energy and, V is potential energy. γ represents the nonconservative and externally force.

If we consider the system without constraints (this means the general coordinate can completely parameterize the configuration space of this system) and external force, the Lagrange equation can be simplify as below:

$$\frac{d}{dt} \left(\frac{\partial}{\partial \dot{g}_k} L \right) - \frac{\partial}{\partial g_k} L = 0 , \tag{4}$$

Example 1: Let us consider the planar pendulum (Figure 9) without external force. We choose θ as the generalized coordinate. This set of general coordinate is independent. This system has no constraints [Ref. 7].

The kinetic energy of the system is:

$$T = \frac{1}{2} mr^2 (\dot{\theta})^2 , \tag{5}$$

and the potential energy is:

$$V = -mgr \cos \theta . \tag{6}$$

The resultant Lagrange operator is:

$$L = T - V$$
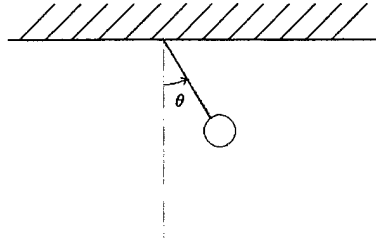
$$= \frac{1}{2} mr^2 \left(\dot{\theta} \right)^2 + mgr \cos \theta \qquad (7)$$

Figure 9. Example 1, Planar Pendulum [Ref. 7]

Substituting Eq. 7 into Eq. 4, the equation of motion of single pendulum is expressed in Eq. 8.

$$\frac{d}{dt}\left(\frac{\partial}{\partial \dot{g}_k} L \right) - \frac{\partial}{\partial g_k} L = \frac{d}{dt}\left(\frac{\partial}{\partial \dot{\theta}} L \right) - \frac{\partial}{\partial \theta} L = 0$$

$$\frac{\partial L}{\partial \dot{\theta}} = \frac{\partial \left(\frac{1}{2} mr^2 \dot{\theta}^2 + mgr \cos \theta \right)}{\partial \dot{\theta}} = mr^2 \dot{\theta}$$

$$\frac{\partial L}{\partial \theta} = \frac{\partial \left(\frac{1}{2} mr^2 \dot{\theta}^2 + mgr \cos \theta \right)}{\partial \theta} = -mgr \sin \theta$$

$$\frac{d}{dt}\left(mr^2 \dot{\theta} \right) = m2r\dot{r} \dot{\theta} + mr^2 \ddot{\theta} = mr^2 \ddot{\theta}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}} \right) - \frac{dL}{d\theta} = mr^2 \ddot{\theta} + mgr \sin \theta = 0$$

$$\ddot{\theta} + \frac{g}{\ell} \sin \theta = 0 \qquad (8)$$

13

The system of Nomad 200 is with constraints and external force. It will be given in Chapter III, Section A, Subsection 1.

C.   LINEARIZATION

The Nomad 200 dynamic model is a nonlinear system (which will be discussed in Chapter III , Section A, Subsection 2). For its control, we use a linearization method. The whole idea of linearization method is to linearize the nonlinear model and apply a well known linear control method. Generally, linearization has two approaches: linear approximation, and feedback linearization. These two approaches are entirely different. The linear approximation is just a linear approximate of the original dynamic model and can be used for the local control around the nominal point. Feedback linearization is the transformation of the nonlinear system dynamics into a linear system. This section will discuss feedback linearization.

The simplest way to describe feedback linearization method is to apply an algebraic quantity, which will cancels the system's nonlinear part. This algebraic quantity transfers the original nonlinear system to a linear system. Example 2 illustrates this method [Ref. 3].

Example 2: Consider the control of the level h of fluid in a tank to a specified level h_d. The control input is the flow u into the tank, and the initial level is h_o. The dynamic model of this tank is:

$$\frac{d}{dt}\left[\int_0^h A(h)dh\right] = u(t) - a\sqrt{2gh} \ , \tag{9}$$

where $A(h)$ is the cross section of the tank and a is the cross section of the outlet pipe. If the initial level h_0 is quite different from the desired level h_d, the control of u involves a nonlinear regulation problem.
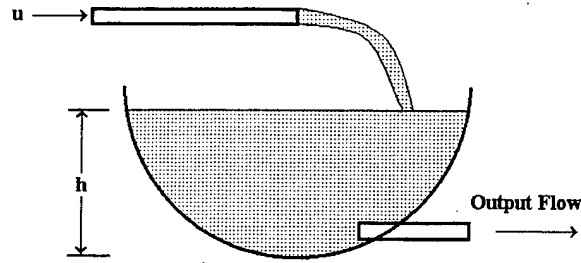
Figure 10. Example 2, Fluid Level

The dynamics can be rewritten as:

$$A(h)\dot{h} = u - a\sqrt{2gh} \ . \tag{10}$$

If $u(t)$ is chosen as:

$$u(t) = a\sqrt{2gh} + A(h)V , \tag{11}$$

with V being an "equivalent input" to be specified, the resulting dynamics is linear:

$$\dot{h} = V \ . \tag{12}$$

Choosing V as:

$$V = \dot{h}_d - \alpha e_h \ , \tag{13}$$

with $e_h = h(t) - hd$ being the level error, and α being a strictly positive constant, the resulting closed loop dynamics is:

$$\dot{e}_h + \alpha \cdot e_h = 0$$

Using Laplace transform, we get: $\Rightarrow \quad se_h + e_h(0^-) + \alpha \cdot e_h = 0$

Then, applying inverse transform: $\Rightarrow \quad e_h = \dfrac{e_h(0^-)}{s + \alpha} \Rightarrow e_h = e_h(0^-) \cdot Exp(-\alpha \cdot t) \tag{14}$

This implies that $e_h \to 0$ as $t \to \infty$.

There is another situation where the system output y is indirectly related to the input u. How is the above feedback linearization method applied? One obvious way is to find a direct connection between these two variables. It can be formally shown that for any

15

controllable system of order n, it will take at most n differentiation of any output for the control input to appear. If no control input appears after n differentiation then the system is uncontrollable [Ref. 3].

Example 3: Consider a third order system [Ref. 3].

$$\begin{aligned}
\dot{x}_1 &= \sin x_2 + \left(x_2 + 1\right)x_3 \\
\dot{x}_2 &= x_1^5 + x_3 \\
\dot{x}_3 &= x_1^2 + u \\
y &= x_1
\end{aligned} \tag{15}$$

To generate a direct relationship between the output y and the input u, we take the differentiation of output y.

$$\ddot{y} = \left(x_2 + 1\right)u + \left(x_1^5 + x_3\right)\left(x_3 + \cos x_2\right) + \left(x_2 + 1\right)x_1^2 \quad . \tag{16}$$

If we let:

$$f(x) = \left(x_1^5 + x_3\right)\left(x_3 + \cos x_2\right) + \left(x_2 + 1\right)x_1^2 , \tag{17}$$

then Eq. (16) can be rewritten as:

$$\ddot{y} = \left(x_2 + 1\right)u + f(x) \quad . \tag{18}$$

Choosing the control input as:

$$u = \frac{1}{x_2 + 1}\left(V - f_1\right) \Rightarrow \ddot{y} = V , \tag{19}$$

and letting the new input be:

$$v = \ddot{y}_d - k_1 e - k_2 \dot{e} , \quad e = y - y_d , \tag{20}$$

then

$$\ddot{e} + k_2 \dot{e} + k_1 e = 0 \quad . \tag{21}$$

If $e(0) = \dot{e}(0) = 0$, then $e(t) \equiv 0, \forall t \geq 0$ i.e., perfect tracking is achieved. Otherwise, *e(t)* converges to zero exponentially.

There is one more situation that should be included with the use of feedback linearization. This situation is the internal dynamics problem. The so-called 'internal dynamics' is the state variable which is unobservable in the input and output equation. If the

16

order of differentiation is not equal to the order of system states, this control model is based on an incomplete system. The whole system dynamics stability is hinged upon the stability of the internal dynamics. For example, consider system dynamics are:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 + u \\ -u \end{bmatrix}$$

$$y = x_1 . \tag{22}$$

Differentiating output y once, gives us:

$$\dot{y} = \dot{x}_1 = x_2 + u , \tag{23}$$

By choosing $u = \dot{y}_d - e - x_2$, the system becomes: $\dot{e} + e = 0$. This will give us a stable system. However, the internal dynamics, $\dot{x}_2 = -(\dot{y}_d - e - x_2)$, is not stable. The whole system's stability depends on x_2.

D. POTENTIAL FIELD THEOREM

When dealing with robot's motion planning, Potential Field Method (PFM) is one of the most popular methods in implementing practical motion planning. This method uses the concept of potential energy. Any object having higher potential energy will automatically attempt to move to the lowest energy state. Applying this concept to a robot's motion planning is to set the robot in an environment. This environment is converted by the PFM into an environment of potential distribution. The position, which is away from the goal position, will have a higher potential energy than the position which is close to the goal position. The goal position is the point with the lowest potential energy in this environment. A graphic representation of the PFM environment is given in Figure 11 [Ref. 4].

The core of PFM is a potential function. This potential function is supposed to reflect the local environment status. This environment status makes the robot choose a suitable direction and velocity. Basically, the potential function consists of two parts: the attraction potential energy of goal position and the repulsive potential energy of environment obstacles. PFM combines these two energies. The robot's motion is according to the result of local configuration. That is why the PFM is also called local motion

17

planning. The PFM's main purpose is to enhance the robot's on line avoidance of obstacles in an unknown environment. The disadvantage of the PFM is that it sometimes will be stuck in a local minimum of the potential field other than the goal position. Dealing with the local minimum is one of the significant issues when using PFM to implement motion planning (A Modified Potential Field Method will be discussed in Chapter III, Section B).

The exerting force is derived from taking the gradient of potential function U:

$$\bar{F}(q) = -\bar{\nabla}U(q) \ , \tag{24}$$

where q is the local configuration and $\bar{\nabla}U(q)$ denotes the gradient vector of U at q. In the case of Nomad 200 , q is defined by two coordinate (x, y), and $\bar{\nabla}U$ can be expressed as:

$$\bar{\nabla}U(q) = \begin{bmatrix} \dfrac{\partial U}{\partial x} \\ \dfrac{\partial U}{\partial y} \end{bmatrix} . \tag{25}$$

Combining the attractive potential energy and repulse potential energy, we will have

$$U(q) = U_{att}(q) + U_{rep}(q) \ , \tag{26}$$

while both potential energies should be independent of each other. The two forces from each gradient vector should be $\bar{F}_{att} = -\bar{\nabla}U_{att}$ and $\bar{F}_{rep} = -\bar{\nabla}U_{rep}$, which are called the attractive and repulsive force. We can choose the attractive potential energy in either of the following forms [Ref. 4]:

$$U_{att}(q) = \frac{1}{2}\xi \rho_{goal}^2(q) \ , \tag{27}$$

or

$$U_{att}(q) = \xi \rho_{goal}(q) \ , \tag{28}$$

where ξ represents a positive scaling factor and $\rho_{goal}(q)$ represents the distance from the current position to the goal position i.e. $\| q - q_g \|$. The respective force is:

$$\bar{F}_{att}(q) = -\bar{\nabla}U_{att}(q) = -\xi\rho_{goal}(q)\bar{\nabla}\rho_{goal}(q) = -\xi(q - q_{goal}) \ , \tag{29}$$

18

$$\vec{F}_{att}(q) = -\xi \vec{\nabla} \rho_{goal}(q) = -\xi \left(\frac{q - q_{goal}}{\left\| q - q_{goal} \right\|} \right) . \tag{30}$$

The first type of potential energy, Eq. 27, has the advantage of stability. This stability is from the attraction force, Eq. 29, which is proportional to the distance between the current position and the goal position. This distance will decrease to zero when robot arrives at the goal point. But this potential energy has a disadvantage also. When robot's position is far from the goal position, the resulting force will become vary large. The second type of potential energy, Eq. 28, has the advantage of keeping attractive force, Eq. 30, steady. The disadvantage of this force is not proportional to the distance between the two positions. This force does not tend toward zero when the robot approaches to the goal position. The best way to use these functions is to combine both function's advantages. We can set a specific distance for the robot. If the distance from robot to the goal position is greater than this distance, the robot should adopt the second type of potential energy. Otherwise, the robot should use the first type of potential energy.

Now, let us consider the repulsive potential energy. This potential energy works in the opposite fashion of the attractive potential energy. The closer the robot comes to the obstacles, the larger the repulse potential energy will be. The obstacle's occupant areas have the highest potential energy to the robot. When robot moves away from that area, the potential effect subsides to zero at a specific range. Since there is no need to worry about the repulsive effect beyond a certain distance, potential energy represented by Eq. 31 can be used to generate repulsive potential energy. The associated repulsive force, Eq. 32, is derived from taking the gradient of Eq. 31.

$$U_{rep}(q) = \begin{cases} \dfrac{1}{2} \eta \left(\dfrac{1}{\rho(q)} - \dfrac{1}{\rho_0} \right)^2 & , \text{ if } \rho(q) \leq \rho_0 \\ 0 & , \text{ if } \rho(q) > \rho_0 \end{cases} \tag{31}$$

where η is a positive scaling factor

19

$$\vec{F}_{rep}(q) = -\vec{\nabla} U_{rep}(q) = \begin{cases} \eta \left(\dfrac{1}{\rho(q)} - \dfrac{1}{\rho_0} \right) \dfrac{1}{\rho^2(q)} \vec{\nabla}\rho(q) & , \text{ if } \rho(q) \le \rho_0 \\ 0 & , \text{ if } \rho(q) > \rho_0 \end{cases} \qquad (32)$$

The total repulsive force is the sum of all response points. The overall force is the sum of the repulsive force plus the attractive force. Optimal performance in terms of how close the robot can come to the goal point and how far the robot will stay away from the obstacles is achieved by adjusting the scaling factors.
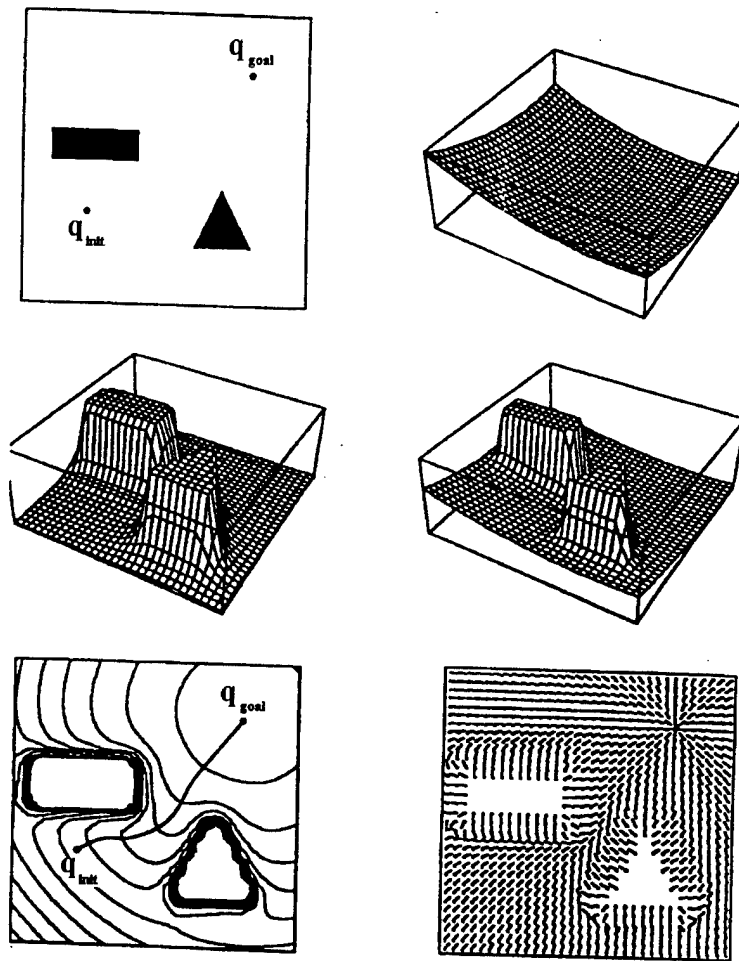

Figure 11. Potential Field

E.   ORIENTATION THEOREM

It is desirable for a robot to move smoothly and quickly. In Section C, the smooth motion is achieved by properly applying a control algorithm. Now, we shall examine how to let the robot move quickly. One of the two ways to solve this problem is to increase moving speed. This method needs to compromise with the system stability. One shouldn't sacrifice the system's stability for fast response. The other way is to find a minimal path for the robot.   We will concentrate on this approach in this section. The relevant research of minimal length path has been done by L. E. Dubins [Ref. 5]. By assuming the robot is a point in the world coordinate, the motion requirements are to move the robot from an initial configuration (position plus orientation) to a desired configuration (position plus orientation) in minimal distance.

Assume the starting point is x_1, and the starting orientation is h_{X1}. The goal point is x_2, and the orientation is h_{X2}. Furthermore, it is assumed that the robot may not turn without moving, and the rotation of the robot is along a fixed circle with radius R. Let n be the dimension of the Euclidean space. Let $C = C(n, x_1, h_{x1}x_2, h_{x2}, R)$ be the collection of all curves X defined on a closed interval [0,L], were $L=L(X)$ varies with X. Proposition one is from Dubin's work. [Ref. 5].

> **Proposition 1.** For any n, x_1, h_{X1} , x_2, h_{X2} and R, there exists an X in $C = C(n, x_1, h_{X1}, x_2, h_{X2}, R)$ of minimal length.

From the set of $\{x_1, h_{X1}, x_2, h_{X2}, R\}$ and Proposition 1, we know that there exists a minimal length path for the motion requirement. This minimal length path is defined as R-geodesic by L. E. Dubins. From the given configuration, we construct two circles which are tangential to the h_{X1} (h_{X2}) in an opposite position with radius R. The left circle from the orientation vector is in counter clockwise direction and the right circle from the orientation is in clockwise direction. Both circles at the starting point and arriving point are the nominal path for the robot to start and arrive. From these four circles, a possible path can be built. Only one of them is the shortest path. The four possible paths are from the x_1's left circle to the x_2's both circles and the x_1's right circle to the x_2's both circles. For the motion direction,

the path always starts from x_1 and follows the nominal path to the side of circle which is in the same movement direction.

Example: The starting point and orientation and the desired goal point and orientation are shown below (Figure 12). Find the minimal path.

From the above discussion, we construct four paths: 1, 2, 3, 4. The length of these four paths can be computed individually by adding two arc length and the length of straight line. The minimal length is then generated. In this case, the path length for Path 1, 2, 3, and 4 are 15.7597, 18.6868, 15.7746 and 15.7587 individually. The minimal length path is Path 4.
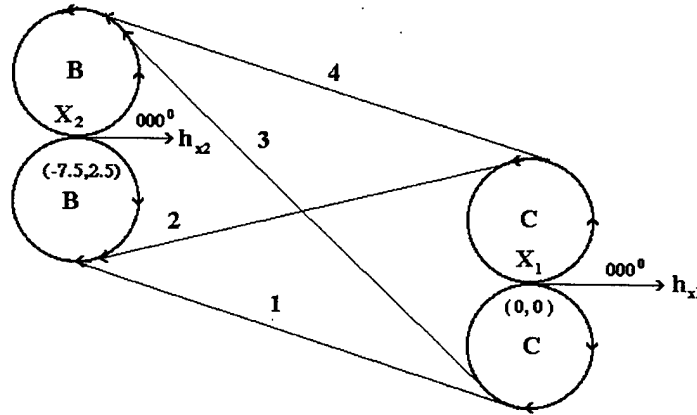
Figure 12. Minimal Length Path

The above method of obtaining the minimal length of a path can also be verified by the L. E. Dubins Proposition.

Proposition 2. If C and B are any two distinct similarly oriented parameterized circles of radius R in a plane, then there exists a unique parameterized straight line segment which leaves C and arrives at B. Furthermore if C and B are oppositely oriented then there exists a parameterized straight line segment which leaves C and arrives at B if and only if no point of B is in the interior of C. If such a segment exists, then it is unique.

Since we are applying this minimal length path to mobile robot, the turning radius is always adjustable (more discussion will be in Chapter III, Section B, Subsection 2). In the Proposition 2, if there have points of *B* in the interior of *C,* we construct the minimum path by varying the radius *R*. We let both circles be exclusive of each other. The minimal length path can be generated by the same procedure. There is another orientation requirement: the robot must be at the same position but facing in the opposite direction. In this case, a third circle needs to be constructed which is tangential to both left and right circles. A path will be constructed from the left or right circle. Follow the nominal path to arrive at the tangent point of third circle. Then shift to the third circle and arrive at the original point but facing the opposite direction (Figure 13).The minimal length path discussion can be concluded with the following theorem:

Theorem 1. Every planar *R*-geodesic is necessarily a continuously differentiable curve which is either (1) an arc of a circle of a radius *R*, followed by a line segment, followed by an arc of a circle of radius *R*; or (2) a sequence of three arcs of circles of radius *R*; or (3) a sub path of a path of type (1) or (2).
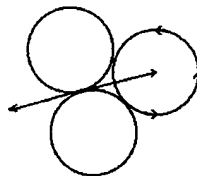

Figure 13. Proposition 2, Minimal Length Path

III.   MOTION PLANNING AND DYNAMIC CONTROL OF THE NOMAD 200

A.   KINEMATICS AND DYNAMIC MODEL OF THE NOMAD 200

1.   Dynamics Model

The dynamics of Nomad 200 depend on its position and orientation. These quantities are not independent, meaning one cannot directly apply the Lagrange method. One must develop the equation of motion when constraints are present. That is, instead of trying to eliminate constraints by a proper choice of general coordinates, which do not exist in this system, we are going to include constraints into the equation of motion.

Nomad 200's Mobile base consists of three wheels which simultaneously steer and rotate. These wheels beneficially simplify this system to a single rolling disk. One may develop the equation of motion from the three wheels. The result would be the same as when one developed it from the single disk. Therefore, we develop the equation of motion from a single disk.

The single disk shown in Figure 14 has two force inputs: τ_ϕ for forward rolling and τ_θ for steering. The rolling angle ϕ and turning angle θ are defined with respect to the vertical reference line and the world coordinates. Based on the Lagrange method we choose the general coordinate as $q=(x, y, \theta, \phi)$.

For the basic development of the motion equation for this disk, one can assume the disk will not slip during rolling. This assumption brings up the constraints equation.

$$\dot{x} - \rho \cos(\theta) \, \dot{\phi} = 0$$
$$\dot{y} - \rho \sin(\theta) \, \dot{\phi} = 0$$

or

$$A(q) \, \dot{q} = \begin{bmatrix} 1 & 0 & 0 & -\rho \cos(\theta) \\ 0 & 1 & 0 & -\rho \sin(\theta) \end{bmatrix} \dot{q} = 0 \, . \tag{33}$$

The number of constraint equations is two and we have four general coordinates. The number of degrees of freedom for this system is equal to two which is consistent with the robot's physical configuration. Now, we can construct the Lagrange operator L:
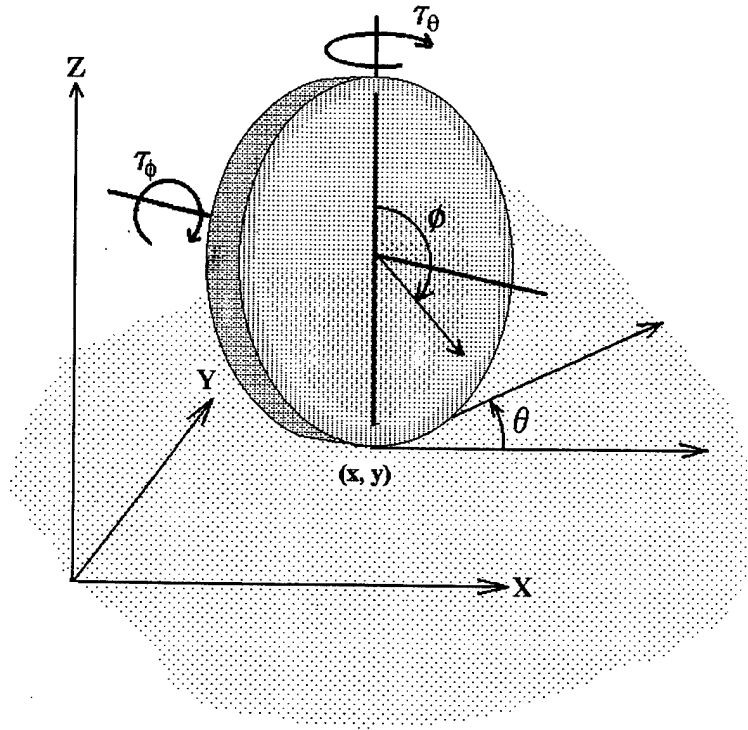
Figure 14. Disk Motion on a Plane

$$K = \frac{1}{2} m (\dot{x}^2 + \dot{y}^2) + \frac{1}{2} (I_1 \, \dot{\theta}^2 + I_2 \, \dot{\phi}^2)$$

$$U = 0$$

$$L(q) = K - U = \frac{1}{2} m (\dot{x}^2 + \dot{y}^2) + \frac{1}{2} (I_1 \, \dot{\theta}^2 + I_2 \, \dot{\phi}^2)$$

(34)

where m is the mass of disk, I_1 & I_2 are the moments of inertia of the disk with respect to the vertical axis and horizontal axis. From the D'Alembert principle

$$\left(A^T(q)\, \lambda\right) \bullet \delta q = 0, \tag{35}$$

and

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} + A^T(q)\, \lambda - \gamma = 0 , \tag{36}$$

where γ represent external applied force. The equation can be written as below:

$$\left(\begin{bmatrix} m & 0 & 0 & 0 \\ 0 & m & 0 & 0 \\ 0 & 0 & I_1 & 0 \\ 0 & 0 & 0 & I_2 \end{bmatrix} \delta q - \begin{bmatrix} 0 \\ 0 \\ \tau_\theta \\ \tau_\phi \end{bmatrix} \right) \bullet \delta q = 0 \tag{37}$$

Expanding Eq. 37, we get:

$$m\,\ddot{x}\,\delta x + m\,\ddot{y}\,\delta y + (I_1\,\ddot{\theta} - \tau_\theta)\,\delta\theta + (I_2\,\ddot{\phi} - \tau_\phi)\,\delta\phi = 0 . \tag{38}$$

From Eq. 33, we have

$$\begin{bmatrix} 1 & 0 & 0 & -\rho\,\cos(\theta) \\ 0 & 1 & 0 & -\rho\,\sin(\theta) \end{bmatrix} \delta q = 0 , \tag{39}$$

and the constraint force is:

$$\begin{aligned} \delta x &= \rho\cos(\theta)\delta\phi \\ \delta y &= \rho\sin(\theta)\delta\phi \end{aligned} \tag{40}$$

$\ddot{x}$ and $\ddot{y}$ can be obtained from differentiating constraint equation, in Eq. 33.

$$\begin{aligned} \ddot{x} &= \rho\cos(\theta)\ddot{\phi} - \rho\sin(\theta)\dot{\phi}\dot{\theta} \\ \ddot{y} &= \rho\sin(\theta)\ddot{\phi} + \rho\cos(\theta)\dot{\phi}\dot{\theta} \end{aligned} \tag{41}$$

Replacing δ_x & δ_y with Eq. 40 and $\ddot{x}$ & $\ddot{y}$ with Eq. 41, the D'Alembert equation can be formed as:

$$m\rho^2\,\ddot{\phi}\,\delta\phi + (I_1\,\ddot{\theta} - \tau_\theta)\,\delta\theta + (I_2\,\ddot{\phi} - \tau_\phi)\,\delta\phi = 0 . \tag{42}$$

Since $\delta\theta$ and $\delta\phi$ are free, we can omit it and the dynamics become:

$$(I_2 + m\,\rho^2)\,\ddot{\phi} + I_1\,\ddot{\theta} - \tau_\theta - \tau_\phi = 0 . \tag{43}$$

27

This equation of motion expresses the system of Nomad 200 as a second order differential equation. The robot's position, *(x, y)*, is related to the rolling and steering angles. Once we identify these angles, the coordinate of Nomad can be derived from:

$$\dot{x} = \rho \, \cos(\theta) \, \dot{\phi}$$
$$\dot{y} = \rho \, \sin(\theta) \, \dot{\phi} \qquad (44)$$

2. System Control

From the last section we have the dynamics expression of the Nomad 200 and system output. Now we can combine these two and rewrite these equations in a state space form as:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \end{bmatrix} = \begin{bmatrix} x_3 \\ x_4 \\ u_1/I_1 \\ u_2/(I_2 + m\rho^2) \\ \rho \cos(x_1)\, x_4 \\ \rho \sin(x_1)\, x_4 \end{bmatrix} \qquad (45)$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_5 \\ x_6 \end{bmatrix}$$

For the convenience of adopting the traditional notation, x_1 denotes the orientation of the disk θ, with respect to the world coordinate, x_2 denotes the rotation angle ϕ, with respect to vertical reference line, x_3 represents $\dot{\theta}$, x_4 represents $\dot{\phi}$, x_5 is the x coordinate, and x_6 is the y coordinate. The system input is τ_θ and τ_ϕ which are represented by u_1 and u_2.

Before we start applying the control algorithm on this system, there is one thing that must be pointed out. If we adopt the system output as above, from the research of Yamamoto and Yun [Ref. 8], this system has been proven to be uncontrollable. The solution for this is to adopt the looking-ahead method. It turns out that the system output should be modified as below:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_5 + L \cos(x_1) \\ x_6 + L \sin(x_1) \end{bmatrix}, \qquad (46)$$

28

where L is a look-ahead distance.

From the Chapter II, Section C, we know if the output variable is indirectly connected to the input variable, we should differentiate the system output so that it is not greater than the system input. Differentiating once yields:

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} \dot{x}_5 - L \sin(x_1)\, \dot{x}_1 \\ \dot{x}_6 + L \cos(x_1)\, \dot{x}_1 \end{bmatrix}$$
$$= \begin{bmatrix} \rho \cos(x_1)\, x_4 - L \sin(x_1)\, x_3 \\ \rho \sin(x_1)\, x_4 + L \cos(x_1)\, x_3 \end{bmatrix} \tag{47}$$

and twice:

$$\begin{bmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{bmatrix} = \begin{bmatrix} \rho \cos(x_1)\, \dot{x}_4 - L \sin(x_1)\, \dot{x}_3 - \rho \sin(x_1)\, x_4\, x_3 - L \cos(x_1)\, x_3^2 \\ \rho \sin(x_1)\, \dot{x}_4 + L \cos(x_1)\, \dot{x}_3 + \rho \cos(x_1)\, x_4\, x_3 - L \sin(x_1)\, x_3^2 \end{bmatrix} . \tag{48}$$

Replacing $\dot{x}_3$ and $\dot{x}_4$, we finally have

$$\begin{bmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{bmatrix} = \begin{bmatrix} \dfrac{-L \sin(x_1)}{I_1} & \dfrac{\rho \cos(x_1)}{I_2 + m\, \rho^2} \\[2ex] \dfrac{L \cos(x_1)}{I_1} & \dfrac{\rho \sin(x_1)}{I_2 + m\, \rho^2} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} -\rho \sin(x_1)\, x_4\, x_3 - L \cos(x_1)\, x_3^2 \\ \rho \cos(x_1)\, x_4\, x_3 - L \sin(x_1)\, x_3^2 \end{bmatrix} . \tag{49}$$

Let

$$[D] = \begin{bmatrix} \dfrac{-L \sin(x_1)}{I_1} & \dfrac{\rho \cos(x_1)}{I_2 + m\, \rho^2} \\[2ex] \dfrac{L \cos(x_1)}{I_1} & \dfrac{\rho \sin(x_1)}{I_2 + m\, \rho^2} \end{bmatrix}$$
$$[E] = \begin{bmatrix} -\rho \sin(x_1)\, x_4\, x_3 - L \cos(x_1)\, x_3^2 \\ \rho \cos(x_1)\, x_4\, x_3 - L \sin(x_1)\, x_3^2 \end{bmatrix} \tag{50}$$

The above equation can be rewritten as

$$\begin{bmatrix} \ddot{y}_1 \\ \ddot{y}_2 \end{bmatrix} = [D] \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + [E] . \tag{51}$$

If we choose the form:

$$u = [D]^{-1} \left(-[E] - k_p \left([Y] - [Y_d] \right) - k_v \left([\dot{Y}] - [\dot{Y}_d] \right) + [\ddot{Y}_d] \right), \tag{52}$$

and

$$[e] = [Y] - [Y_d] .$$ (53)

The original system becomes

$$[\ddot{e}] + k_v \cdot [\dot{e}] + k_p \cdot [e] = 0 .$$ (54)

This has resulted in a linear second order system. Now we can apply the linear control algorithm. Properly choosing k_v and k_p as below will result in the desired response.

$$k_v = 2 \, \xi \, \omega_n$$
$$k_p = \omega_n^2$$ (55)

where ω_n is the undamped natural frequency, ξ is the damping ratio.

B.   MOTION PLANNING OF THE NOMAD 200

1.   Modified Potential Field Implementation

From Chapter II, Section D, we already know the potential field method (PFM). In this section, we are going to discuss the implementation of PFM and modify PFM to avoid local minimal points and obstacles in a laboratory environment. The local minimal point is the point other than the goal point which is surrounded by higher potential energy field. When the robot moves into this area, it will be stuck.

For the discussion of the implementation of PFM, we need to recall the configuration of Nomad 200. Nomad 200 itself has two sets of coordinate systems: world coordinate system and body-fixed coordinate system. The position information from integrating wheel rotation is based on world coordinates. The robot's position was initially at point (0,0), when robot is turned on or set to zero, and the facing direction of robot is 000 degree. This coordinate is the same as a Cartesian coordinate. The robot will keep track of its position relative to that point. The sonar sensor, infrared sensor and tactile sensor of Nomad 200 are set up on the body-fixed coordinate system. For the computation based on the same coordinate, we need to transform one coordinate to the other. In this thesis we transform the body frame system to the world coordinate system. The transformation matrix is given below.

30

$$T = \begin{bmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{bmatrix} \qquad (56)$$

Detection range is different for sonar, infrared and tactile sensor. The sonar sensor was chosen as the main sensor for evaluation of the PFM. The infrared sensor can be used to plan motions near obstacles. The return value from the 16 sonar sensors are used as $\rho(q)$ in Eq. 31 of Chapter II, Section D. This gives us the repulsive force of the local environment. The attractive force is computed from the difference of the distance between goal position and current position. These computations are implemented in c language on the host computer. Part of this implement is shown in Figure 15.

```
F_att_rob[0] = xi * D_att_rob[0];
F_att_rob[1] = xi * D_att_rob[1];

for (i = 0; i <= 15; i++)
    {
      rho_float = (double) (SonarRange[i]);
      if (rho_float < rho_0)
      {
          F_rep[0] +=
          -eta * (1.0 / rho_float-1.0/rho_0) * cos((double)(i) * 0.392699) / (rho_float);

          F_rep[1] +=
          -eta * (1.0 / rho_float-1.0/rho_0) * sin((double)(i) * 0.392699) / (rho_float);
      }
    }
F_tol[0] = F_att_rob[0] + F_rep[0];
F_tol[1] = F_att_rob[1] + F_rep[1];
```

Figure 15. Source Code for Attractive and Repulsive Force Implementation

Where F_att_rob[0] (or F_att_rob[1]) denotes the attractive force in x (or y) direction. F_rep[0] (or F_rep[1]) denotes the repulsive force from obstacles in x (or y) direction. The 'for' loop is to retrieve the values from 16 sonar sensors. The overall combined force is calculated in the last two lines.

One must adjust the positive scaling factor ξ and η that are denoted as *xi* and *eta* in Figure 15. To get the optimal motion approach, one must also decide how far the robot can move to the obstacles and how large the attractive potential should put on the robot. All these considerations are interrelated. If the scaling factor for attractive force is increased, this may sometime result in a potential energy large enough to push a small obstacle, like a stool or chair. If the repulsive scaling factor is increased it may make the robot behave as if it were scared of obstacles and stand excessively far from obstacles. So, the optimal scaling factor depends on the environment.

Finally, the local minimum problem must be addressed. Based on the laboratory environment, a typical robot's environment can be demonstrated by the cases shown in Figure 16. In these situations, if the potential field method is applied that was developed in the previous section, the robot will be stuck at the local minimum point and cannot get out by using the same motion strategy. A modified potential field method is developed to overcome this situation.
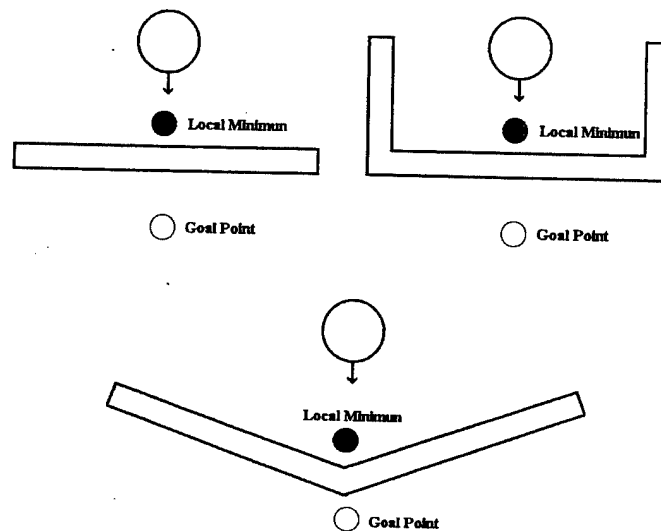

Figure 16. Simplified Laboratory Environment

32

If we consider the geometry of these configurations, it is not hard to discover a common phenomenon. This common phenomenon is that the distance between the local minimal point and goal position is shorter compared to the other points on the robot's side of the barrier. Any other points on the robot side will be greater than this distance and will increase gradually until arriving at a corner point. From the corner point, the distance, between the goal position and the robot position, will start decreasing. We can use this configuration to enhance the potential field method. The algorithm for this modified potential field method is shown below:

```
main()
{
initial robot;  get goal point;
        while (not arrive at goal point)
            {
                calculated local potential field;
                if (stuck on a local minimum)
                    {
                        while (the distance to the goal point is increasing)
                            {
                                follow the wall of the obstacle;
                            }
                    }
                use PFM approach
            }
}
```

2.   Orientation Implementation

From the previous discussion on the shortest path and considering the robot orientation, one must build two circles and a straight line. The circle is associated with robot

position and the goal position. The straight line is used to connect these two circles. Based on this strategy and the physical configuration, we will have four circles and four straight lines associated with the robot position and the goal position. Two circles touch the orientation vector on both sides of robot position. The other two circles touch the orientation vector on both sides of the goal position. And the four straight lines are to connect these four circles which always leave from the robot circle and arrive at the goal circle (Figure 17).
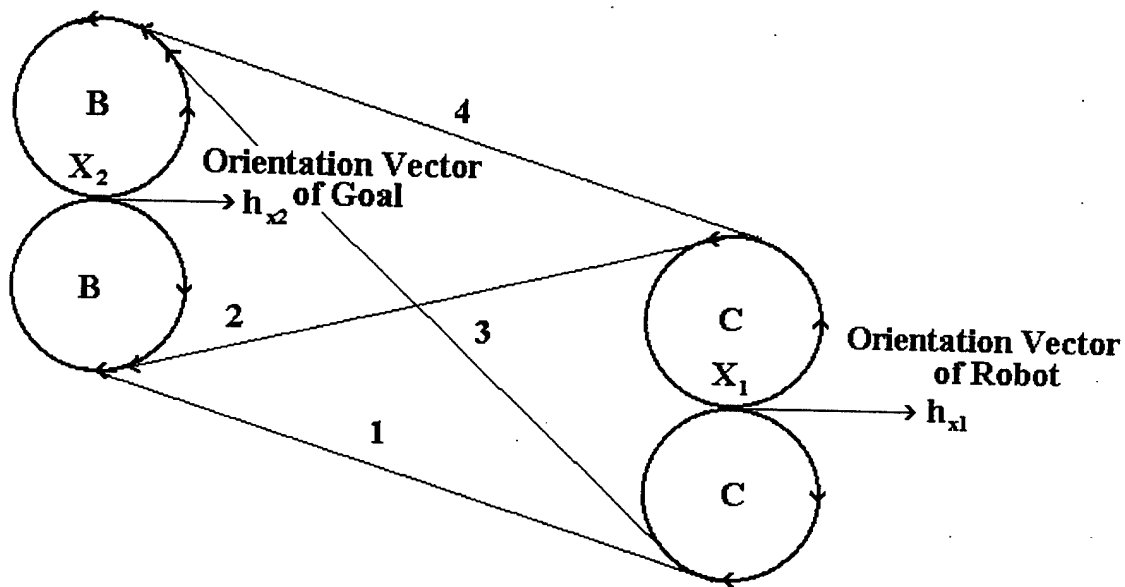
Figure 17. Minimal Length Path

Only one of the combination of the two circles and one straight line provides the shortest path for robot. The path length can be calculated according to its geometric configuration. Once we get the shortest path, we can generate a set of sub-goal point. The last sub-goal point will be coincident with the goal point. The first sub-goal point is the robot's starting point. These sub-goal points are given to the robot one at a time until the robot approaches the sub-goal point to within a specific range.

34

The turning radius for a moving vehicle should be adjusted according to the vehicle velocity. Based on the control algorithm of the Nomad 200, this algorithm makes Nomad 200 an asymptotically stable system. The turning radius is not critical for Nomad 200. The approach velocity is automatically decreased when robot comes close to the goal point or sub goal point. And considering the laboratory environment, the robot turning radius should be adjusted to fit the compact moving environment. For an obstacle free environment, the shortest path was computed and shown in Figure 18. In Figure 18, the goal point was given in the four quadroon with respect to robot. The points marked by * are transition points connecting a circle with a line.
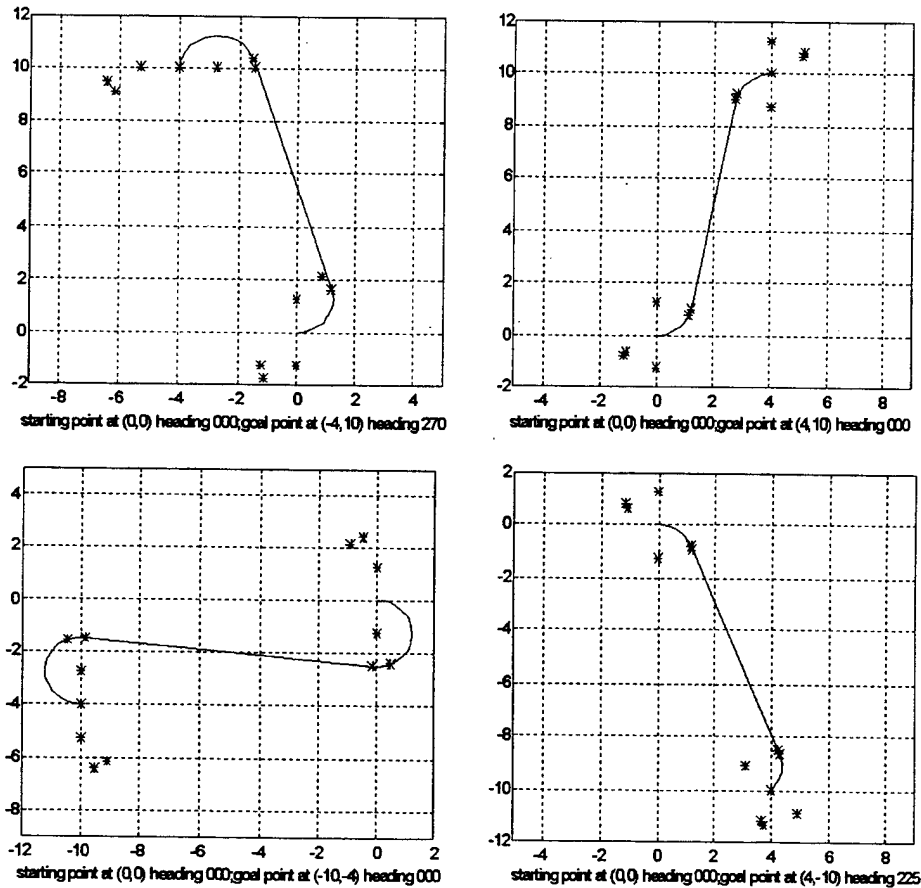

Figure 18. Shortest Path for Robot to Move

35

C. SIMULATION AND PHYSICAL EXPERIMENT

1. Simulation Using Matlab

Based on the dynamics of Nomad 200, a simulation of this dynamic system using Simulink toolbox of Matlab is made. The entire system includes two separate blocks: a goal point generating block and a system dynamics block. Figure 19 shows this composition.
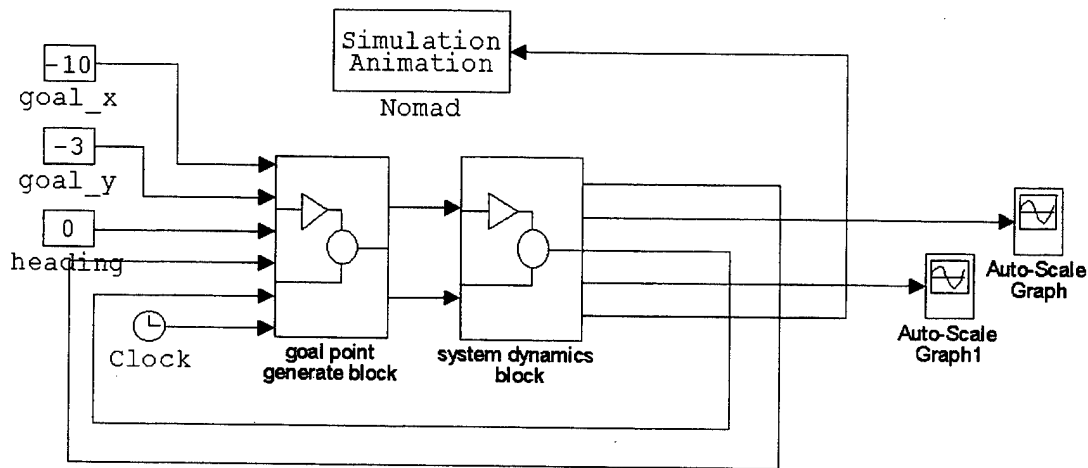
Figure 19. The Simulation Group

For this system, the input variable is the x position, y position, and heading of the goal configuration. The output variable is the actual x position and y position. The animation block shows simulated robot movements.

Figure 20 is the goal point generation block. It takes the specific goal point and orientation and puts them into the *nomadp06* function. The result is a path point matrix. Based on input from Channels 4 & 5, this function can determine when to output the next path point to the dynamics block. The clock and feedback *Fcn* blocks at channel 6 & 7 are used for the inside counter.

The system dynamics block includes an equation of motion for the Nomad 200 and a linearization feedback block. Inside the FB1.m function, the feedback quantity is derived from Chapter III, Section A, Subsection 2. The closed loop input to this block is sequential

36

path point x & y. The closed loop output is the robot's actual position. The feedback terms are the robot's instantaneous position and $\theta, \dot{\theta}, \phi$. There is a connection to the animation block which displays simulated robot motion. The simulated results are shown in Figure 22.
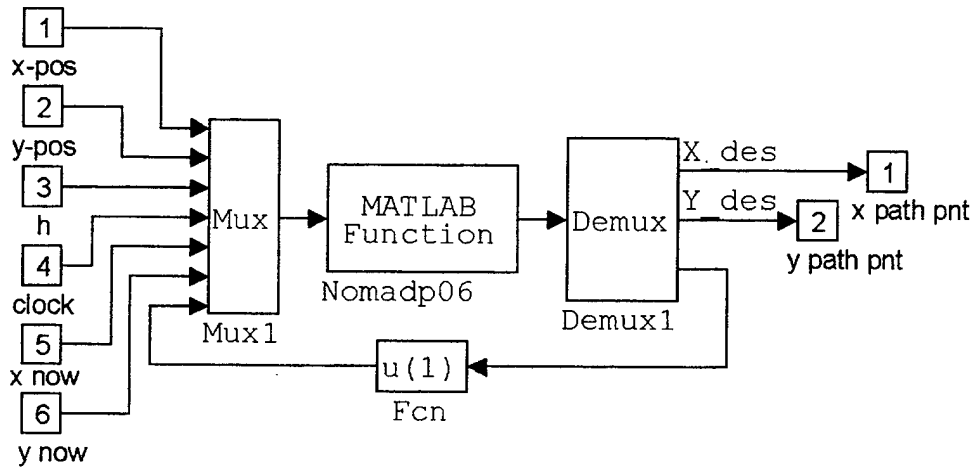
Figure 20. The Path Point Generation Block
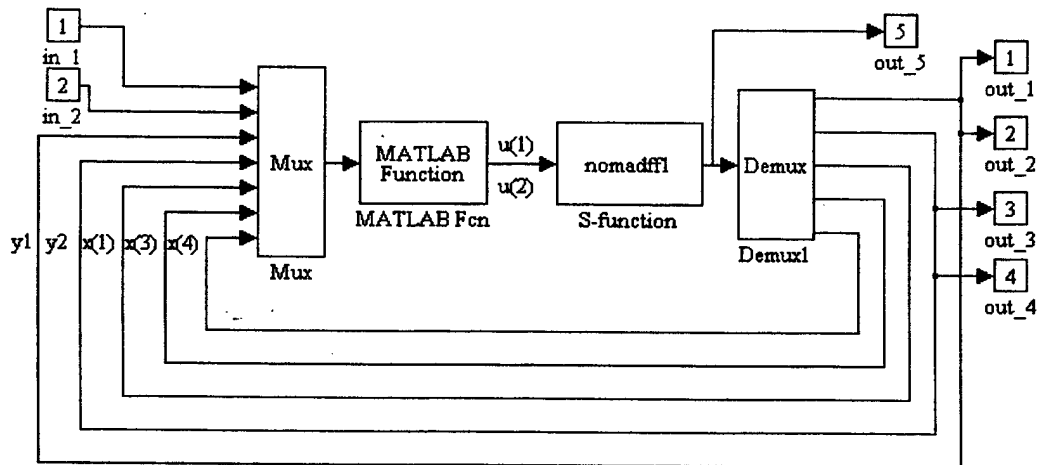
Figure 21. The System Dynamics Block
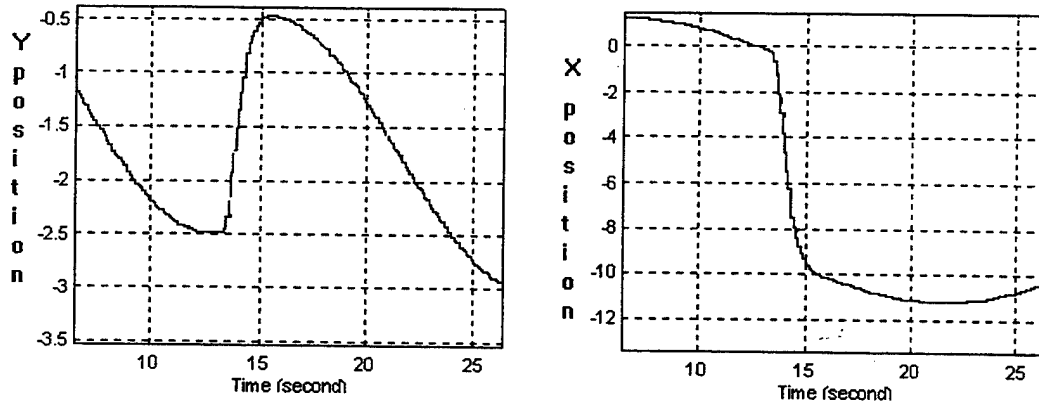
Figure 22. Simulated Result

2. Simulation Using Nomad 200 Robot Simulator

The simulation using Matlab does not simulate the potential field method for avoiding obstacles. This can be completed by the Nomad 200 simulation program. Figure 23 to Figure 25 shows the moving situation in a simplified laboratory environments. It is clearly demonstrated that a robot at a local minimal point remain stationary for a short period of time while it is working to determine if it is stuck or not. When the "stuck check" is passed, the robot will automatically switch to the "follow wall subroutine". When the distance between robot's position and goal position is decreasing, robot will automatically shift back to the PFM and approach to the goal point. During the final approach period, the robot will check its surround environment to determine if there is enough space to smoothly turn and arrive at the goal point.

38
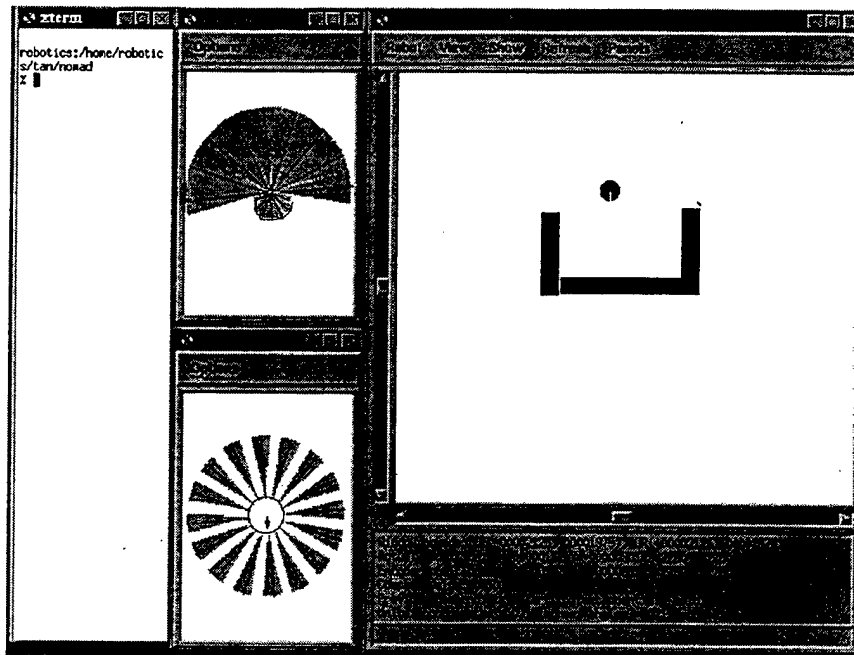
Figure 23. Starting Figure of Simulated Program
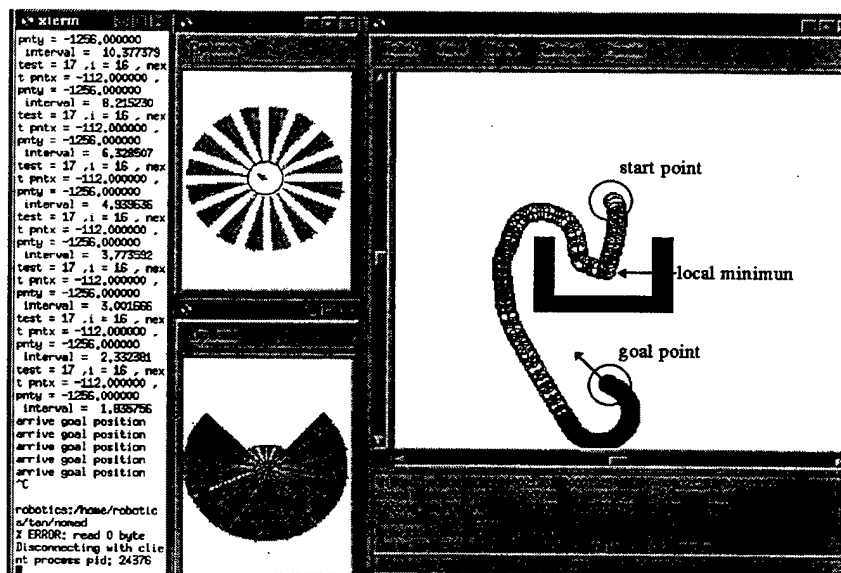
Figure 24. The Simplified 1st Lab Environment Situation

Figure 25. The Simplified 2nd Lab Environment Situation

Figure 26. The Simplified 3rd Lab Environment Situation

40
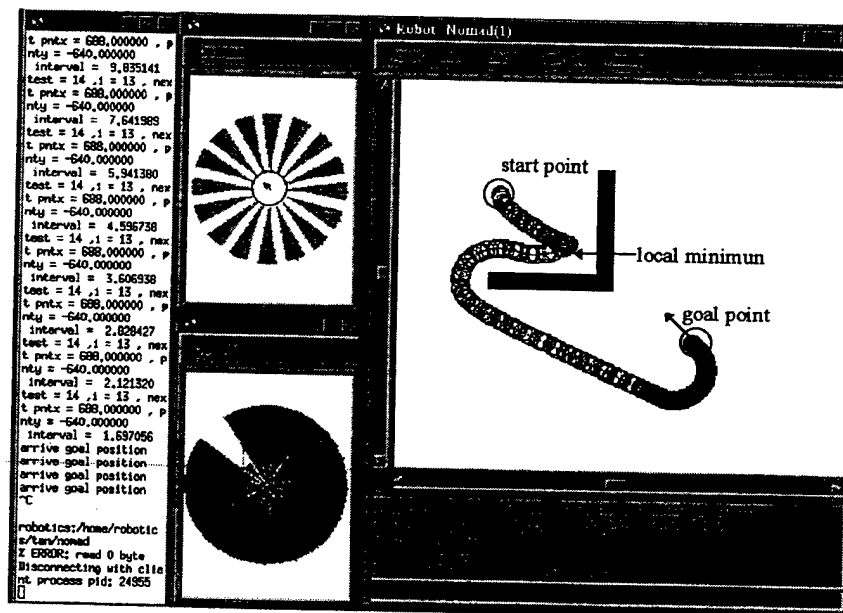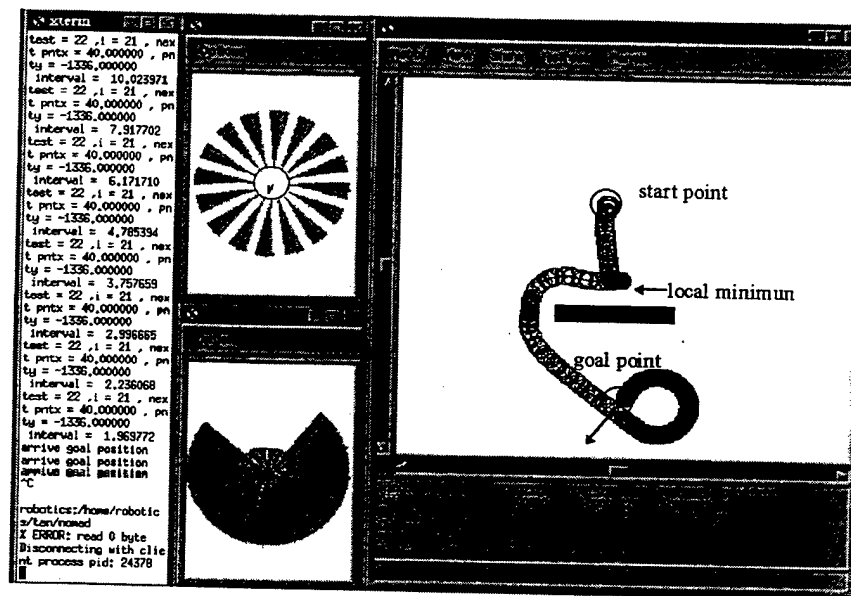
3. Physical Experiment

The physical experiment was conducted in Bullard Hall Rm 201. Two types of experiment were done: one is using PFM with a local minimal other than the goal point, the other is using PFM without local minimal except goal point. Both experiments demonstrated (Figure 27 and Figure 28) that the robot has the ability to get to the goal position and face in a specified orientation. The first experiment uses a modified PFM, which allows the robot to avoid local minima. The robot starts at point A. When the robot freezes at point B, it switches from the PFM to the follow wall sub-routine. Until the distance between robot position and goal point decreases at point C, the robot shifts back to the PFM and approaches the goal position. In the mean time, the robot routinely checks the path point. If a clear path is shown, the robot will go directly to the goal position following a nominal path.

The second experiment is similar to the first one. Since there are no other local minima except the goal position in the environment, the robot will move to the goal position using PFM. For both experiments, the final approaching step is implemented using Dubin's method. These results from physical experiments are consistent with the theoretical results.
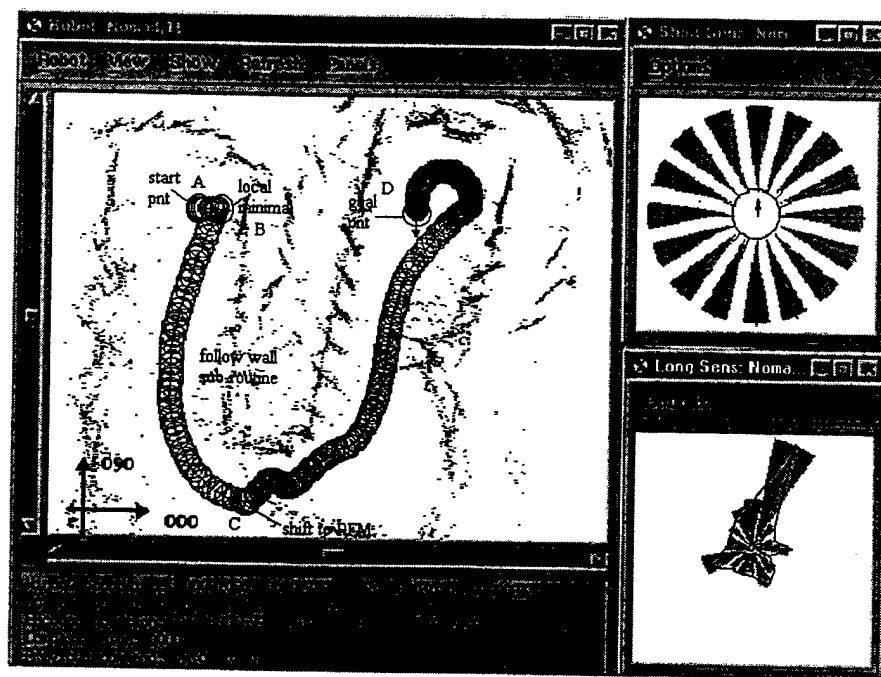
Figure 27. Using PFM With a Local Minimum Other Than the Goal Point
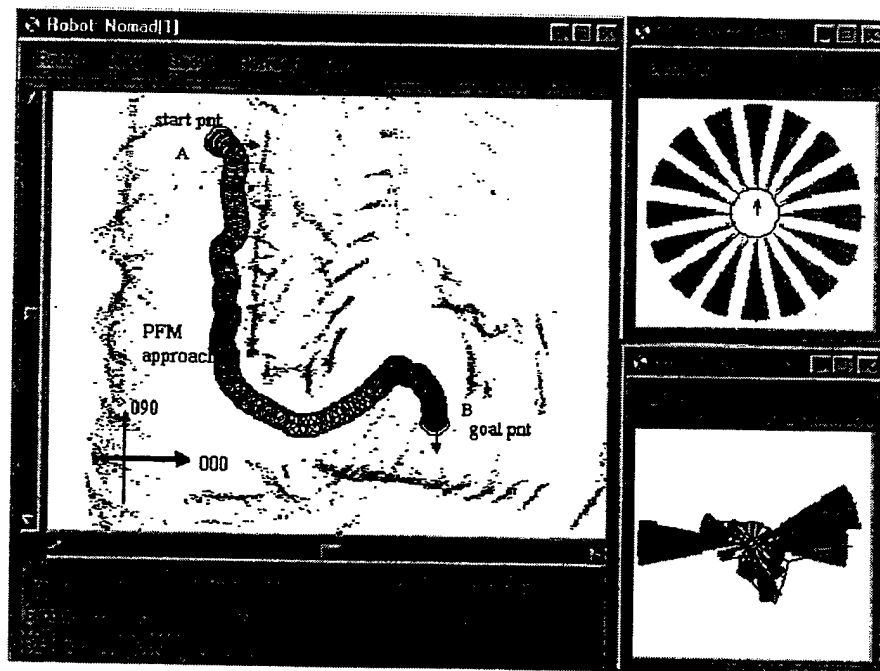
Figure 28. Using PFM without a Local Minimum Except the Goal Point

IV. CONCLUSION

There are three fundamental questions in mobile robot research: (i) where am I; (ii) where am I going; and (iii) how will I go there. This thesis focuses on the third question. Thus it is assumed that a goal configuration (position and orientation) of the mobile robot is given.

The first objective is to find a path from the current location of the mobile robot to the goal configuration in the presence of unknown obstacles. This is the motion planning problem. The second objective is to actually control the robot to move to its goal configuration. This is the feedback control problem.

In this thesis, a dynamic model for the Nomad 200 mobile robot is developed using the Lagrange equation with constraints. The nonlinear dynamic model is linearized by a nonlinear state feedback, which transforms the original system dynamics into a second order linear system. A linear feedback is further designed to satisfy performance requirements. The feedback control algorithms are verified by Matlab simulations.

A motion planning algorithm based on the potential field method is implemented to find a path to the goal configuration. Since the original potential field algorithm has a local minimum problem, a modified potential field algorithm is developed. The modified algorithm is able to overcome local minima in a typical laboratory environment. The modified algorithm is also combined with Dubin's algorithm in order for the robot to smoothly reach the goal orientation.

The proposed algorithms have been simulated using Matlab and the Nomad Robot Simulator, and have been implemented on a Nomad 200 Mobile Robot. Results from simulations and physical experiments show that the algorithms are effective.

Recommendations for future work include further improvement of the potential field algorithm. The modified algorithm presented in this thesis works well in a typical laboratory environment. It may not work in a more complicated environment such as a maze.

APPENDIX A. SOURCE CODE FOR MATLAB

```
function [gxy]=nomadp06(u)
% u(1)=desire x position
% u(2)=desire y position
% u(3)=desire orientation
% u(4)=current x position
% u(5)=current y position
% u(6)=clock
% u(7)=delay feed back


% generate the whole needed path point
[pp]=nomadp03(u);

% get the number of total path point
Num=size(pp);

% set the minimum distant criteria
% as long as the robot close to this range
% this matlab function will sent out next point.
d=0.1;

% main loop
if u(6) < 0.1,% clock time < 0.1
   u(7)=2;% point to next path point,
         % 1st point is the robot starting point.
   gxy(3)=2;
   gxy(1)=pp(1,2);gxy(2)=pp(2,2);
end

% check how close the current position to the previous poing
% if less then specfic distance ,send out the next path point
if sqrt( ( u(4)-pp(1,u(7)) )^2 +( u(5)-pp(2,u(7)) )^2 ) < d,
         % if the point to the last point then stop.
         if u(7)==(Num(1,2)-1),
            pause;
         end
         % put next point as goal point
         gxy(1)=pp(1,u(7)+1);
         gxy(2)=pp(2,u(7)+1);
         gxy(3)=u(7)+1;
```

45

```
        gxy(3)=u(7)+1;

end

if sqrt( ( u(4)-pp(1,u(7)) )^2 +( u(5)-pp(2,u(7)) )^2 ) > d,
        % if checking distance is larger than specific deistance
        % keep the same value
        gxy(1)=pp(1,u(7));
        gxy(2)=pp(2,u(7));
        gxy(3)=u(7);

end
```

```
function [pp]=nomadp03(u)
%
% pp:given a mtx vector of x & y to plant
%
% u(1):receive x from commander
% u(2):receive y from commander
% u(3):receive heading from commander
%
%
% the data structure of PD(path data):
% +-----+------+------+------+------+------+------+------+
% |     |start |begin |stop  |end   |begin |stop  |min # |
% |     |center|point |point |center|point |point |radius|
% +-----+------+------+------+------+------+------+------+
% | X   |(1,1) |(1,2) |(1,3) |(1,4) |(1,5) |(1,6) |(1,7) |
% +-----+------+------+------+------+------+------+------+
% | Y   |(2,1) |(2,2) |(2,3) |(2,4) |(2,5) |(2,6) |(2,7) |
% +-----+------+------+------+------+------+------+------+
%
% the data structure of Pn(tangent point):
% +-----+------+------+------+------+
% |     |L->L  |L->R  |R->L  |R->R  |
% |     |circle|circle|circle|circle|
% +-----+------+------+------+------+
% | X   |(1,1) |(1,2) |(1,3) |(1,4) |
% +-----+------+------+------+------+
% | Y   |(2,1) |(2,2) |(2,3) |(2,4) |
% +-----+------+------+------+------+
% |goalX|(1,1) |(1,2) |(1,3) |(1,4) |
% +-----+------+------+------+------+
% |goalY|(2,1) |(2,2) |(2,3) |(2,4) |
% +-----+------+------+------+------+

x=u(1);
y=u(2);
ang=u(3);

% set rotation radius R = 1.25
        R = 1.25;
        goal_ang=atan2(y,x);

        gRcir_ang=ang-(pi/2);
```

47

```
        gLcir_ang=ang+(pi/2);


% the goal's right and left circle center coordinate
        gRx=x+R*cos(gRcir_ang);
        gRy=y+R*sin(gRcir_ang);

        gLx=x+R*cos(gLcir_ang);
        gLy=y+R*sin(gLcir_ang);


% the oringinal's left and right circle center coordinate
        oLx=0;oRx=0;
        oLy=R;oRy=-R;


%*******distant and slope between these circle center******************
        dis_ang(1,1)=((gLx-oLx)^2 +(gLy-oLy)^2 )^0.5;% oLgL dis
        dis_ang(1,2)=atan2((gLy-oLy),(gLx-oLx));% oLgL ang

        dis_ang(2,1)=((gRx-oLx)^2 +(gRy-oLy)^2 )^0.5;%oLgR dis
        dis_ang(2,2)=atan2((gRy-oLy),(gRx-oLx));%oLgR ang

        dis_ang(3,1)=((gLx-oRx)^2 +(gLy-oRy)^2 )^0.5;%oRgL dis
        dis_ang(3,2)=atan2((gLy-oRy),(gLx-oRx));%oRgL ang

        dis_ang(4,1)=((gRx-oRx)^2 +(gRy-oRy)^2 )^0.5;%oRgR dis
        dis_ang(4,2)=atan2((gRy-oRy),(gRx-oRx));%oRgR ang

        % the short loop here is to convert the domain of atan2
        % from pi->-pi to 0->2pi . reason is I want to calculate
        % arc length S=r*theta ,theta should always be positive.
        for KK=1:4,
            [dis_ang(KK,2)]=checkit(dis_ang(KK,2));
        end

%*******find the tangential point**************************************
        % left circle to left circle, tangential point on the
        %original is x=pn(1,1) y=pn(2,1),
        %tangential point on the goal is x=pn(3,1) y=pn(4,1)

        pn(1,1)=oLx+R*cos(dis_ang(1,2)- pi/2);
        pn(2,1)=oLy+R*sin(dis_ang(1,2)- pi/2);

        pn(3,1)=gLx+R*cos(dis_ang(1,2)- pi/2);
```

48

```
        pn(4,1)=gLy+R*sin(dis_ang(1,2)- pi/2);

% left circle to right circle, tangential point on the
%original is x=pn(1,2) y=pn(2,2),
%tangential point on the goal is x=pn(3,2) y=pn(4,2)

theta1=asin(2*R/dis_ang(2,1));

pn(1,2)=oLx+R*cos(dis_ang(2,2)- pi/2 + theta1);
pn(2,2)=oLy+R*sin(dis_ang(2,2)- pi/2 + theta1);

pn(3,2)=gRx+R*cos(dis_ang(2,2)+ pi/2 + theta1);
pn(4,2)=gRy+R*sin(dis_ang(2,2)+ pi/2 + theta1);


% right circle to left circle, tangential point on the
%original is x=pn(1,3) y=pn(2,3),
%tangential point on the goal is x=pn(3,3) y=pn(4,3)

theta2=asin(2*R/dis_ang(3,1));

pn(1,3)=oRx+R*cos(dis_ang(3,2)+ pi/2 - theta2);
pn(2,3)=oRy+R*sin(dis_ang(3,2)+ pi/2 - theta2);

pn(3,3)=gLx+R*cos(dis_ang(3,2)- pi/2 - theta2);
pn(4,3)=gLy+R*sin(dis_ang(3,2)- pi/2 - theta2);

% right circle to right circle, tangential point on the
%original is x=pn(1,4) y=pn(2,4),
%tangential point on the goal is x=pn(3,4) y=pn(4,4)

pn(1,4)=oRx+R*cos(dis_ang(4,2)+ pi/2);
pn(2,4)=oRy+R*sin(dis_ang(4,2)+ pi/2);

pn(3,4)=gRx+R*cos(dis_ang(4,2)+ pi/2);
pn(4,4)=gRy+R*sin(dis_ang(4,2)+ pi/2);


% find the shortest path

% path 1 is from left circle to left circle
        cird=[gLx pn(3,1) x 2;gLy pn(4,1) y 0];
```

```matlab
        [dif]=angdiff(cird);
    path(1)=R*(dis_ang(1,2))+...
        ((pn(1,1)-pn(3,1))^2 + (pn(2,1)-pn(4,1))^2 )^0.5 +...
        R*dif(1);
% path 2 is from left circle to right circle
        cird=[gRx pn(3,2) x 1;gRy pn(4,2) y 0];
        [dif]=angdiff(cird);
    path(2)=R*(dis_ang(2,2)+theta1)+...
        ((pn(1,2)-pn(3,2))^2 +(pn(2,2)-pn(4,2))^2 )^0.5 +...
        R*dif(1);
% path 3 is from right circle to left circle
        cird=[gLx pn(3,3) x 2;gLy pn(4,3) y 0];
        [dif]=angdiff(cird);
    path(3)=R*(2*pi-dis_ang(3,2)+theta2)+...
        ((pn(1,3)-pn(3,3))^2 +(pn(2,3)-pn(4,3))^2 )^0.5 +...
        R*dif(1);
% path 4 is from right circle to right circle
        cird=[gRx pn(3,4) x 1;gRy pn(4,4) y 0];
        [dif]=angdiff(cird);
    path(4)=R*(2*pi-dis_ang(4,2))+...
        ((pn(1,4)-pn(3,4))^2 +(pn(2,4)-pn(4,4))^2 )^0.5 +...
        R*dif(1);
% find out the shortest path
min=1;
        for K=1:4,
                if path(K)<path(min),
                  min=K;
                end
        end


% generate path point for the shortest

if min==1,
pd=[oLx 0 pn(1,1) gLx pn(3,1) x 1;
   oLy 0 pn(2,1) gLy pn(4,1) y R];
end

if min==2,
pd=[oLx 0 pn(1,2) gRx pn(3,2) x 2;
   oLy 0 pn(2,2) gRy pn(4,2) y R];
end
```

```
if min==3,
pd=[oRx 0 pn(1,3) gLx pn(3,3) x 3;
   oRy 0 pn(2,3) gLy pn(4,3) y R];
end

if min==4,
pd=[oRx 0 pn(1,4) gRx pn(3,4) x 4;
   oRy 0 pn(2,4) gRy pn(4,4) y R];
end


[pp]=pathpnt1(pd);
```

```
function [pp]=pathpnt1(pd)
%
% pp->path point,
% pd->path data,is 2 by 7 mtx
% this function is a subroutine for nomadp01.m
% the purpose of this code is to find out the path
% point along the starting circle and ending circle
%
%************************************************************
d=15;
    if pd(1,7) < 3,
        cird=[pd(:,1) pd(:,2) pd(:,3) [2;0]];
        a=1;
        [diff]=angdiff(cird);

        while (a*d*pi/180) < diff(1),
            ppx1(a)=pd(1,1)+pd(2,7)*cos(diff(2)+a*(d*pi/180));
            ppy1(a)=pd(2,1)+pd(2,7)*sin(diff(2)+a*(d*pi/180));
            a=a+1;
        end
    end

    if pd(1,7) > 2,
        cird=[pd(:,1) pd(:,2) pd(:,3) [1;0]];
        a=1;
        [diff]=angdiff(cird);

        while (a*d*pi/180) < diff(1),
            ppx1(a)=pd(1,1)+pd(2,7)*cos(diff(2)-a*(d*pi/180));
            ppy1(a)=pd(2,1)+pd(2,7)*sin(diff(2)-a*(d*pi/180));
            a=a+1;
        end
    end
%************************************************************

    if (pd(1,7)==1) | (pd(1,7)==3),
        cird=[pd(:,4) pd(:,5) pd(:,6) [2;0]];
        a=1;
        [diff]=angdiff(cird);

        while (a*d*pi/180) < diff(1),
            ppx2(a)=pd(1,4)+pd(2,7)*cos(diff(2)+a*(d*pi/180));
```

52

```
                        ppy2(a)=pd(2,4)+pd(2,7)*sin(diff(2)+a*(d*pi/180));
                        a=a+1;
                end
        end

        if (pd(1,7)==2) | (pd(1,7)==4),
                cird=[pd(:,4) pd(:,5) pd(:,6) [1;0]];
                a=1;
                [diff]=angdiff(cird);

                while (a*d*pi/180) < diff(1),
                        ppx2(a)=pd(1,4)+pd(2,7)*cos(diff(2)-a*(d*pi/180));
                        ppy2(a)=pd(2,4)+pd(2,7)*sin(diff(2)-a*(d*pi/180));
                        a=a+1;
                end
        end

pp=[pd(1,2) ppx1 pd(1,3) pd(1,5) ppx2 pd(1,6);
    pd(2,2) ppy1 pd(2,3) pd(2,5) ppy2 pd(2,6)];
```

```
function [diff]=angdiff(cird)
%
% the data structure of cird(circle data):
% +-----+------+-----------+----------+------------------+
% |      |center|begin point|stop point|turning direction |
% +-----+------+-----------+----------+------------------+
% |  X  |(1,1) |(1,2)      |(1,3)     |(1,4)             |
% +-----+------+-----------+----------+------------------+
% |  Y  |(2,1) |(2,2)      |(2,3)     |(2,4)             |
% +-----+------+-----------+----------+------------------+


startang=atan2(cird(2,2)-cird(2,1),cird(1,2)-cird(1,1));
[startang]=checkit(startang);
endang=atan2(cird(2,3)-cird(2,1),cird(1,3)-cird(1,1));
[endang]=checkit(endang);

% 1 represent right turn
if cird(1,4)==1,
   if startang > endang,
      dif=startang-endang;
   else
      dif=2*pi-endang+startang;
   end
end
% 1 represent left turn
if cird(1,4)==2,
   if startang > endang,
      dif=2*pi-startang+endang;
   else
      dif=endang-startang;
   end
end

diff=[dif startang endang];
```

54

```matlab
function [sys, x0] = nomadff1(t,x,u,flag)
%
%      Represents the state-space equations:
%
%           dx/dt = A.x + B.u
%           y = C.x + D.u
%
%      as an M-file.
%
%      Where x is the state vector, u is vector of inputs,
%      and y is the vector of outputs.
%
%

I1=100;I2=50;m=20;ro=0.5;L=1;

if nargin~=4
   if nargin==0
        flag = 0;
   else
        error('??? Wrong number of input arguments.');
   end
end

if abs(flag) == 1    % If flag = 1, return state derivatives, xDot

    sys(1,1) = x(3);
    sys(1,2) = x(4);
    sys(1,3) = u(1)/I1;
    sys(1,4) = u(2)/(I2+m*ro^2);
    sys(1,5) = ro*cos(x(1))*x(4);
    sys(1,6) = ro*sin(x(1))*x(4);

    x0=x;
elseif flag == 3    % If flag = 3, return system outputs, y

    sys(1,1) = x(5)+L*cos(x(1));
    sys(1,2) = x(6)+L*sin(x(1));
    sys(1,3) = x(1);
    sys(1,4) = x(3);
    sys(1,5) = x(4);
```

55

```
    x0=x;
elseif flag == 0    % If flag = 0, return initial condition data, sizes and x0
    sys = [6; 0; 5; 2; 0;0];
    x0 = [0 0 0 0 0 0]';

else
    % If flag is anything else, no need to return anything
    % since this is a continuous system
    sys = [];

end
```

APPENDIX B. SOURCE CODE FOR NOMAD 200 ROBOT SIMULATOR

```
/**/
/* program name:Nomad1.c*/
/* written by Ko-cheng Tan*/
/**/
/* The purpose of this program is to utilize the potential field */
/* method on approaching the assigned position and orientation.*/
/* */
/* The above function is completed by nine sub-routine which was*/
/* called by main() routine.*/
/* The function of each of these sub-routine is summaried below:*/
/* Movement():dealing with potential field generation on the*/
/*          local enviroment and consequently generate moving*/
/*          velocity and direction.*/
/* GetSensorData():update the sensor variable from the global*/
/*             variable, state[].*/
/* Movement1():main sub-routine of godirect() program*/
/* sign():is a small program which check the result value, */
/*          return -1 if a negative result was checked, otherwise,*/
/*      return 1.*/
/* *pathpnt():this is the sub-routine which computed the path */
/*          point for the robot to follow. At final, the robot */
/*          will face the assignment orientation.*/
/* *angdiff():this program computed the angle difference in the*/
/*          evaluate of path point and the choose of shortest*/
/*          path.*/
/* checkit():this program check the angle magnitude, keep the*/
/*          angle in the range (0 to 2pi).*/
/* clearchk():this program working on the final approaching */
/*          period, to check the goal position vicinity is */
/*          clear for the robot to make a smooth turn.*/
/* godirect():this program working after the clearchk(), if the*/
/*          return value is positive then the robot will go*/
/*          direct to the goal position.*/


#include "Nclient.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```c
#define TRUE 1
#define FALSE 0
#define PI 3.1415

void GetSensorData(void);
void Movement(void);
void Movement1(void);
int sign(int);

double *pathpnt (double *);
double *angdiff (double *);
void    checkit (double *);
int     clearchk(double *,double *);
void    godirect(void  );


long SonarRange[16];  /* array of sonar readings (inches) */
long IRRange[16];  /* array of infrared readings (no units) */
long robot_config[4],goal_config[4];

double O_att[2],OO_att[2],pathp[50],diffff[3];
float xOld, yOld, x, y;

int BumperHit = 0;  /* boolean value */
int count,check,first;

main(unsigned int argc, char** argv)
{
 int i,index;int oldx,oldy,oldsteering,oldturret;int order[16];
 double X,Y;

 SERV_TCP_PORT = 7705;
 strcpy(ROBOT_MACHINE_NAME, "nomad");
 /* name of the robot (Ndirect) */

 connect_robot(1);  init_mask();  conf_tm(1);

 xOld=0.0; yOld=0.0; x=0.0; y=0.0;/* this part initial */
 count=0; check=1; first=1;    /* the variable value.*/
       for (i = 0; i < 2; i++)  /* for the double type*/
           { O_att[i] = 0.0;    /* variable, value should*/
```

```c
        OO_att[i] = 0.0;}   /* be set as 0.0*/

    for (i = 0; i < 16; i++)/* configure the sonar sensor */
        order[i] = i;  /* firing order.*/
    conf_sn(1,order);        /* */

    for (i = 0; i < 16; i++)/* configure the infared */
    order[i] = i;      /* sensor firing order.*/
        conf_ir(1,order);        /* */

tk("Start to test poential field.");
printf("Please select a goal configuration. \n");
get_robot_conf(goal_config);

printf("%10d %10d %10d\n",goal_config[0],goal_config[1],
        goal_config[2]);

    X =(double) goal_config[0];
    Y =(double) goal_config[1];

while (!BumperHit)            /* Main loop. */
        {                            /* if the robot */
        GetSensorData();          /* was not hitten*/
          if ( clearchk( &X,&Y) )/* on bumper, the*/
                {                        /* robot should keep*/
              printf(" check goal position clear \n");
              godirect();/* going, in the mean time, get */
              }            /* sensor data,*/
        Movement();      /* decide moving*/
        }/* direction and velocity until the clear check*/
        /* on the goal point return clear signal.*/
  disconnect_robot(1);
}

void Movement (void)
{
  double xi=5.0;
  double rho_att=100.0;
  double rho_0=2000.0;
  double eta=40000.0;
  double D_att[2];
  double D_att_rob[2];
```

```c
    double dist,H1,H2,H3,Ss1,Ss2,Ss3;
    double F_att[2],F_att_rob[2],F_rep[2],F_tol[2];
    double robot_steering_angle,phi;
    double rho_float,sidesteer;

    int i,tests;
    int panic;
    int tvel, svel, dirdiff,dirnow,dirwant;
    int rho, k;

    dirnow=0; /* keep current direction.*/
    dirwant=0;/* computed desired direction.*/

    panic = FALSE;
    for (i = 12; i <= 15; i++)
/* if checking range < 8 && 10, set panic */
        if (SonarRange[i] < 8 && IRRange[i] < 10) panic = TRUE;
    for (i = 0; i <= 4; i++)
/* the above checking is based on the fron */
        if (SonarRange[i] < 8 && IRRange[i] < 10) panic = TRUE;
                                                    /* sensor*/
        if (check) /* if check = 1 , mean no stock */
          {/* this subroutine is brifing below.*/
          /* 1.save the current position.*/
          /* 2.check the position movement using hypot()*/
          /* 3.if stuck, increase 'count'*/
          /* 4.if 'count' > 2, adopt follow wall procedure*/
          /* 5.otherwise, save current position as old value*/
          /*   waiting next stuck check.*/

          x = State[34];
          y = State[35];
              if (hypot(x-xOld,y-yOld)<1.0)
                {
                count = ++ count;
                printf(" count = %d \n",count);
                    if ( count == 2 )
                      {
                      check=0;
                      count=0;
                      first=1;
                      }
```

```
        }
        xOld = x;
        yOld = y;
    /* below showing the implement of potential field method*/
    /* 1.computed the attraction force by a transformation.*/

        D_att[0]=(double)(goal_config[0]-robot_config[0]);
            /* x component: distant to the goal position */
        D_att[1]=(double)(goal_config[1]-robot_config[1]);
            /* y component: distant to the goal position */

        robot_steering_angle =
            ((double)robot_config[2])*PI/(10.0*180.0);
        phi=robot_steering_angle;

D_att_rob[0] = cos(phi)*D_att[0] + sin(phi)*D_att[1];
D_att_rob[1] = -sin(phi)*D_att[0] + cos(phi)*D_att[1];
    /* D_att_rob is the attraction force related to the*/
    /* body fixed fram*/
dist = hypot(D_att_rob[0],D_att_rob[1]);
    if (dist < rho_att)
        {
            F_att_rob[0] = xi*D_att_rob[0];
            F_att_rob[1] = xi*D_att_rob[1];
        }
    else
        {
            F_att_rob[0] = xi*rho_att*D_att_rob[0]/dist;
            F_att_rob[1] = xi*rho_att*D_att_rob[1]/dist;
        }
    /* this part is the two kind of attraction force, */
    /* state in the charter two section D, we set a*/
    /* specific distance to select one of this two */
    /* attraction force.*/

F_rep[0] = 0.0;
F_rep[1] = 0.0;

for (i = 0; i <= 15; i++)
    {
        rho_float = (double) (SonarRange[i]);
            if (rho_float < rho_0)
```

```
                    {
                    F_rep[0] += -eta*(1.0/rho_float - 1.0/rho_0)
                            *cos((double)(i) * 0.392699)/(rho_float);
                    F_rep[1] += -eta*(1.0/rho_float - 1.0/rho_0)
                            *sin((double)(i) * 0.392699)/(rho_float);
                    }
                }

        F_tol[0] = F_att_rob[0] + F_rep[0];
        F_tol[1] = F_att_rob[1] + F_rep[1];

        tvel = (int) (0.1 * F_tol[0]);

        svel = (int) (200.0*sin(atan2(F_tol[1],F_tol[0])));

        vm(tvel,svel,svel);

        }
        else    /* the rest of here is dealing stuck situation   */
        {       /* the follow wall program is also implement here */
          D_att[0] = (double)(goal_config[0]-robot_config[0]);
          D_att[1] = (double)(goal_config[1]-robot_config[1]);

        /* the loop below is checking when transfer from potential */
/* method to follow wall method, is it first time excute */
    /* follow wall prog . if it is then do a right turn before*/
    /* doing the other. otherwise just do follow wall prog .  */

        if ( first )
          {/* this part doing the follow wall preparation*/
          /* by turn the heading 100 degree relative to */
          /* goal position.*/

          dirdiff=200;
          while(!(abs(dirdiff)<100))
                  {
                  tvel=0;
                  svel=dirdiff/5;
                  vm(tvel,svel,svel);
                  gs();

                  robot_config[2] = State[36];
```

62

```c
        robot_steering_angle =
                ((double)robot_config[2])*PI/(10.0*180.0);
        phi = robot_steering_angle;

        D_att_rob[0] =  cos(phi)*D_att[0] + sin(phi)*D_att[1];
        D_att_rob[1] = -sin(phi)*D_att[0] + cos(phi)*D_att[1];

        dirnow=(int) (atan2(D_att_rob[1],D_att_rob[0])*1800.0/PI)
                +((int) robot_config[2]);
        dirnow=dirnow-1100;
        dirdiff=dirnow-((int) robot_config[2]);
        }

  first=0;
        for (i = 0; i < 2; i++)
          {
          O_att[i] = 0.0;
  OO_att[i] = 0.0;
          }
}

if (!panic)
  tvel = 80;   /* can be between 0 and 280 */
else
  tvel = 0;

if (hypot(D_att[0],D_att[1]) < (hypot(O_att[0],O_att[1])-10.0))
    check=1;

OO_att[0]=O_att[0];
OO_att[1]=O_att[1];

O_att[0]=D_att[0];
O_att[1]=D_att[1];

H1 =((double) SonarRange[15]) * cos(2.0*PI*15.0/16.0);
H2 =((double) SonarRange[0] ) * cos(2.0*PI*0.0 /16.0);
H3 =((double) SonarRange[1] ) * cos(2.0*PI*1.0 /16.0);
/* H1,H2,H3 checking the heading range for follow wall routine.*/
/* if any one of these value is < 20 then do a quick turning.*/

svel=0;sidesteer=0.0;
```

```c
        if ((H1<20.0)||(H2<20.0)||(H3<20.0))
      {svel = -65;tvel = 5;}
        else
          svel=0;


      Ss1 =((double) SonarRange[2]) * sin(2.0*PI*2.0/16.0);
      Ss2 =((double) SonarRange[3]) * sin(2.0*PI*3.0/16.0);
      Ss3 =((double) SonarRange[4]) * sin(2.0*PI*4.0/16.0);
      /* Ss1,Ss2,Ss3 checking the side range for follow wall routine.*/
      /* set the distant to the wall as 120, any combine of side range*/
      /* will affect the steering factor. Otherwise, not change.*/

      sidesteer = Ss1 + Ss2 + Ss3;

      svel += (int)(sidesteer-120.0);
      tests = abs(svel);

      /* avoid the transition too fast, set limit to 100. */
      if (tests < 100)
        svel=svel;
      else
        svel=svel*20/36;

      vm(tvel,svel,svel);


      }


}



/* GetSensorData(). Read in sensor data and load into arrays. */
void GetSensorData (void)
{
 int i;

 gs();

 for (i = 0; i < 16; i++)
  {
   SonarRange[i] = State[17+i];
   IRRange[i] = State[1+i];
```

64

```c
        }

  for (i = 0; i < 4; i++)
    robot_config[i] = State[34+i];

  if (State[33] > 0)
    {
      BumperHit = 1;
      tk("Ouch.");
      printf("Bumper hit!\n");
    }
}




int sign(int x)
{
  return x>0?1:-1;
}




void godirect(void)
{

  double PX,PY,PH,X,Y,ang,gRcir_ang,gLcir_ang; /* initial value */
  double gRx,gRy,gLx,gLy; /* goal R & L circle center coordinate */
  double dis_ang[4],cird[8];
  double pn[4],pathR,pathL;    /* the tangential pnt */
  double *dif,*pp,pd[10];
  double  R,interval,dis_angT,degree,argx,argy;
  double Gx,Gy,X1,Y1,slope;
  int min,i,allset,noright,noleft;
  int tvel,svel,absd,test,count1;
  i = 0;

        PX=(double) robot_config[0];
        PY=(double) robot_config[1];
        PH=(double) robot_config[2];

        X1 =(double) goal_config[0];
        Y1 =(double) goal_config[1];
```

```
ang =(double) goal_config[2];

    slope = (ang+1800.0)*PI/1800.0;
    X = X1 + 100.0 * cos( slope );
    Y = Y1 + 100.0 * sin( slope );

    R = 300.0; /* robot radius */
    interval = hypot(PX-X,PY-Y);
if ( interval < 1200.0 )
    R = interval / 4.0;

    printf("R = %2f \n",R);

    gRcir_ang=ang-900.0;  /* 900 = pi/2 */
    gLcir_ang=ang+900.0;  /* 900 = pi/2 */

/* calculate the left and right circle center coordinate */
    gRx = X + R* cos( gRcir_ang*PI / 1800.0 );
    gRy = Y + R* sin( gRcir_ang*PI / 1800.0 );
    gLx = X + R* cos( gLcir_ang*PI / 1800.0 );
    gLy = Y + R* sin( gLcir_ang*PI / 1800.0 );

/* distance and slope between position now and pnt above */
            dis_angT = atan2(PY-gRy,PX-gRx);
            checkit( &dis_angT );
    dis_ang[0] = dis_angT;
            dis_angT = atan2(PY-gLy,PX-gLx);
            checkit( &dis_angT );
    dis_ang[1] = dis_angT;

    dis_ang[2] = hypot(gRy-PY,gRx-PX);

    dis_ang[3] = hypot(gLy-PY,gLx-PX);
/* find the tangential pnt */
    /* tangential pnt at right circle */
    pn[0] = gRx + R*cos(dis_ang[0] - PI/2);
    pn[1] = gRy + R*sin(dis_ang[0] - PI/2);
    /* tangential pnt at left circle */
    pn[2] = gLx + R*cos(dis_ang[1] + PI/2);
    pn[3] = gLy + R*sin(dis_ang[1] + PI/2);

/* find the shortest path */
```

```
/* calculate the arc length first */
cird[0]=gRx;cird[1]=pn[0];cird[2]=X;cird[3]=1;
cird[4]=gRy;cird[5]=pn[1];cird[6]=Y;cird[7]=0;
dif = angdiff( cird );

pathR = R * dif[0] + dis_ang[2];

cird[0]=gLx;cird[1]=pn[2];cird[2]=X;cird[3]=2;
cird[4]=gLy;cird[5]=pn[3];cird[6]=Y;cird[7]=0;
dif = angdiff( cird );

pathL = R * dif[0] + dis_ang[3];

if ( pathR < pathL )
        min = 1;
else
        min = 2;


/* generate path pnt for the shortest one */
/* for min = 1 , right circle ,the path point should be */

allset = 1;

while ( !(allset > 2) )
        {
        if ( min == 1 )
                {/* pd stand for path data , include point position now,*/
                 goal's right circle center coordinate, starting point,
                 end point which is goal position, radius,
                 turning direction.                     */

                 pd[0]=PX;pd[1]=gRx;pd[2]=pn[0];pd[3]=X;pd[4]=1.0;
                 pd[5]=PY;pd[6]=gRy;pd[7]=pn[1];pd[8]=Y;pd[9]=R;


                /* function pathpnt generate point at path according*/
                 to the path data.                     */

                 pp=pathpnt(pd);

        test=(int) *(pp+49);
                        /* this value is the total number of path point */
```

```c
        count1 = 0;

/* noright = 1 mean when doing clear check on right circle*/
   path there have point the robot can not pass. noright = 0
   mean all the path is clear                    */

   noright = 0;

/* this loop check every point on the path */
   while ( count1 < test )
       {
           argx=*(pp+count1);argy=*(pp+count1+25);
           if ( !clearchk ( &argx,&argy ) )
             noright = 1; /* point not clear */
           count1 = ++ count1;
       }

/* if after check every point is clear */
   if ( ! noright )
     {
     i = 0;
     X1 = (double) goal_config[0];
     Y1 = (double) goal_config[1];

     while( !(i == test) )
           {
           GetSensorData();

           PX=(double) robot_config[0];
           PY=(double) robot_config[1];
           PH=(double) robot_config[2];

        Gx = *(pp+i);
           Gy = *(pp+i+25);

             printf("test = %d ,i = %d , next pntx = %2f , pnty = %2f
\n",test,i,Gx,Gy);

           interval = hypot( Gy-PY , Gx-PX ) / 10.0;

           printf(" interval = %2f \n",interval);
```

```c
        if ( interval < 2.0 )
             i = ++ i;

        test=(int) *(pp+49);

        goal_config[0]=(long) Gx;
        goal_config[1]=(long) Gy;

        Movement1();
        }
        goal_config[0] = (long) X1;
        goal_config[1] = (long) Y1;

        while(1)
          {printf("arrive goal position \n");

             vm(0,0,0);}
        }
    else
        {
             min = 2;
             allset = ++ allset;
        }
    }

/* for min = 2 , left circle , the path point should be */
if ( min == 2 )
        {/* pd stand for path data , include point position now,*/
          goal's left circle center coordinate, starting point,
          end point which is goal position, radius,
          turning direction.                            */

          pd[0]=PX;pd[1]=gLx;pd[2]=pn[2];pd[3]=X;pd[4]=2.0;
          pd[5]=PY;pd[6]=gLy;pd[7]=pn[3];pd[8]=Y;pd[9]=R;


        /* function pathpnt generate point at path according*/
          to the path data.                        */

          pp=pathpnt(pd);

test=(int) *(pp+49);
```

```c
        count1 = 0;

        /* noleft = 1 mean when doing clear check on left circle*/
        path there have point the robot can not pass. noleft = 0
        mean all the path is clear                  */

        noleft = 0;

        /* this loop check every point on the path */
        while ( count1 < test )
            {
                argx=*(pp+count1);argy=*(pp+count1+25);
                        if ( !clearchk ( &argx,&argy ) )
                            noleft = 1; /* point not clear */
                count1 = ++ count1;
                }


        /* if after check every point is clear */
        if ( ! noleft )
          {
          i = 0;
          X1 = (double) goal_config[0];
          Y1 = (double) goal_config[1];
        test=(int) *(pp+49);
                while( !(i == test))
                    {
                    GetSensorData();

                    PX=(double) robot_config[0];
                    PY=(double) robot_config[1];
                    PH=(double) robot_config[2];

                    Gx = *(pp+i);
                    Gy = *(pp+i+25);

                        printf("test = %d ,i = %d , next pntx = %2f , pnty = %2f
\n",test,i,Gx,Gy);

                    interval = hypot( Gy-PY , Gx-PX ) / 10.0;

                    printf(" interval = %2f \n",interval);
```

70

```
                        if ( interval < 2.0 )
                            i = ++ i;

                        test=(int) *(pp+49);

                        goal_config[0]=(long) Gx;
                        goal_config[1]=(long) Gy;

                        Movement1();

                        }
                            goal_config[0] = (long) X1;
                            goal_config[1] = (long) Y1;

                            while(1)
                            {
                            printf("arrive goal position \n");
                            vm(0,0,0);
                            }
                    }
            else
                    {
                    min = 1;
                    allset = ++ allset;
                    }


                }
    /* end of min = 2 loop */


        }
    /* end of while loop */


    }
.   /* end of main function loop */



int clearchk(double *argx,double *argy)
{
  double Rx,Ry,Dist,Angl,Gx,Gy,Tx,Ty,Rh,S1,S2;
  int S_num,A,B;
```

```c
    Rx = (double) robot_config[0];
    Ry = (double) robot_config[1];
    Rh =((double) robot_config[2])*PI/1800.0;
    Gx = *argx;
    Gy = *argy;


    Tx =  cos(Rh)*(Gx-Rx)+sin(Rh)*(Gy-Ry);
    Ty = -sin(Rh)*(Gx-Rx)+cos(Rh)*(Gy-Ry);

/* the purpose to plus 200.0 here is to to make sure*/
   when checking the wanted point , there have enough
   room to let robot pass by.                         */

    Dist = hypot( Tx , Ty ) + 120.0;
    Angl = atan2( Ty , Tx );
          checkit( &Angl );

    S_num = abs( (int) (Angl * 1800.0 / (225.0 * PI)) );

    A=S_num;
    B=S_num+1;
    if (S_num == 15)
       B=0;
    S1=((double) SonarRange[A])*10.0;S2=((double) SonarRange[B])*10.0;

    if ( ( S1 > Dist ) && ( S2 > Dist ) )
/* if ( S1 > Dist ) */
          return(1);
     else
          return(0);


}



double *angdiff(double *arg)
{
double startang,endang,dif;
double *cird;

cird=arg;
```

```c
        startang = atan2(*(cird+5)-*(cird+4),*(cird+1)-*cird);
        checkit(&startang);
        endang = atan2(*(cird+6)-*(cird+4),*(cird+2)-*cird);
        checkit(&endang);

    if (*(cird+3)==1.0)
        {
        if ( startang > endang )
            dif = startang - endang;
        else
            dif = 2.0*PI - endang + startang;
        }
    if (*(cird+3)==2.0)
        {
        if ( startang > endang )
            dif = 2.0*PI - startang + endang;
        else
            dif = endang - startang;
        }

difff[0] = dif;
difff[1] = startang;
difff[2] = endang;
return(difff);


}




void checkit(double *Deg)
{
 if ( *Deg < 0.0 )
    *Deg += 2.0*PI;
}




double *pathpnt(double *pd)
{
 double a,d,cird[8];
 double *diff;
```

```
int A;

A=0;

for ( A = 0 ; A < 49 ; A++ )
   pathp[A]=0.0;

d=150.0*PI/1800.0;

if (pd[4]==1.0)
       {
               /* pd[0]=PX; pd[1]=gRx; pd[2]=pn[0]; pd[3]=X; pd[4]=1.0;*/
                pd[5]=PY; pd[6]=gRy; pd[7]=pn[1]; pd[8]=Y; pd[9]=R;  */

               cird[0]=*(pd+1); cird[1]=*(pd+2);
               cird[2]=*(pd+3); cird[3]=*(pd+4);
               cird[4]=*(pd+6); cird[5]=*(pd+7);
               cird[6]=*(pd+8); cird[7]=*(pd+9);

               diff=angdiff(cird);
               a = 0.0;A = 0;
               pathp[A]    = *(pd+2);
               pathp[A+25] = *(pd+7);
               A=1;
         while ((a*d) < *diff)
               {
                pathp[A]    = *(pd+1) + *(pd+9) * cos ( *(diff+1) - a*d );
                pathp[A+25] = *(pd+6) + *(pd+9) * sin ( *(diff+1) - a*d );
                a = ++a;A = ++A;
                }
               pathp[A]    = *(pd+3);
               pathp[A+25] = *(pd+8);
               A = ++A;
               pathp[A]    = (double) goal_config[0];
               pathp[A+25] = (double) goal_config[1];
               A = ++A;
               pathp[A]    = (double) goal_config[0];
               pathp[A+25] = (double) goal_config[1];
        }
       else
        {
               /* pd[0]=PX; pd[1]=gLx; pd[2]=pn[2]; pd[3]=X; pd[4]=2.0;*/
```

```
                pd[5]=PY; pd[6]=gLy; pd[7]=pn[3]; pd[8]=Y; pd[9]=R;  */

            cird[0]=*(pd+1); cird[1]=*(pd+2);
            cird[2]=*(pd+3); cird[3]=*(pd+4);
            cird[4]=*(pd+6); cird[5]=*(pd+7);
            cird[6]=*(pd+8); cird[7]=*(pd+9);

            diff=angdiff(cird);
            a = 0.0;A = 0;
            pathp[A]   = *(pd+2);
            pathp[A+25] = *(pd+7);
            A=1;
        while ((a*d) < *diff)
            {
            pathp[A]   = *(pd+1) + *(pd+9) * cos ( *(diff+1) + a*d );
            pathp[A+25] = *(pd+6) + *(pd+9) * sin ( *(diff+1) + a*d );
            a = ++a;A = ++A;
            }
            pathp[A]   = *(pd+3);
            pathp[A+25] = *(pd+8);
            A = ++A;
            pathp[A]   = (double) goal_config[0];
            pathp[A+25] = (double) goal_config[1];
            A = ++A;
            pathp[A]   = (double) goal_config[0];
            pathp[A+25] = (double) goal_config[1];
        }


        A=(double) A;
        pathp[49]=A;
        return(pathp);

    }



void Movement1 (void)
{
double DX,DY,Rx,Ry,roboh;
int tvel,svel;
tvel=0;svel=0;
```

```
roboh=((double)robot_config[2])*PI/1800.0;

DX = (double)(goal_config[0]-robot_config[0]);
DY = (double)(goal_config[1]-robot_config[1]);

Rx =  cos(roboh)*DX + sin(roboh)*DY;
Ry = -sin(roboh)*DX + cos(roboh)*DY;

/* the value 2.0 & 3.0 in tvel , svel equation is simply doing amplify*/
  speed when travel and steering.                            */
    tvel = (int) ( hypot( Ry,Rx ) * 2.0 / 10.0 );
    svel = (int) ( atan2( Ry,Rx ) * 180.0*3.0 / PI );

    if (tvel > 70)
       tvel = 70;

vm(tvel,svel,svel);

}
```

LIST OF REFERENCES

1. *Nomad 200 Mobile Robot User's Guide*, Mountain View, California: Nomadic Technologies, Inc., 1995.

2. Murray, Richard M., Zexiang Li and S. Shankar Sastry, *A Mathematical Induction to Robotic Manipulation*, 1994.

3. Slotine, Jean-Jacques E. and Weiping Li, *Applied Nonlinear Control*, Englewood Cliffs,, New Jersey: Prentice Hall, 1991.

4. Latombe, Jean-Claude, *Robot Motion Planning*, Norwell, MA: Kluver Academic Publisher, 1991.

5. Dubins, L. E., "On Curves of Minimal Length, with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents." *American Journal of Mathematics,* vol. 79, 1957, pp. 497-516.

6. *Nomad 200 Mobile Robot Language Reference Manual*, Mountain View, California: Nomadic Technologies, Inc., 1995.

7. Agrawal, Brij, "Dynamics and Control of Smart Structures," Class Notes, Dept. of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, California, 1995.

8. Yamamoto, Yoshio, Xiaoping Yun, "Coordinating Locomotion and Manipulation of a Mobile Manipulator," *IEEE Transactions on Automatic Control,* vol. 39, no. 6, June 1994.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center 8725 John J. Kingman Rd., STE 0944 Ft. Belvoir, VA 22060-6218	2
2.	Dudley Knox Library Naval Postgraduate School 411 Dyer Rd. Monterey, California 93943-5101	2
3.	Professor Daniel Collins Code AA/CO Naval Postgraduate School Monterey, California 93943-5101	1
4.	Professor Xiaoping Yun Code EC/YX Naval Postgraduate School Monterey, California 93943-5101	2
5.	Professor Isaac Kaminer Code AA/KA Naval Postgraduate School Monterey, California 93943-5101	1
6.	Lcdr Ko-Cheng Tan 3F #140-2 Yee-Hsing ST. Xiao-Kung Ward Kaohsiung TAIWAN R.O.C.	2